ATARI USER'S GUIDE

BASIC and Graphics for Atari 400 600XL 800 800XL and 1200XL

Mark Ellis, Robert Ellis, and Larry Joel Goldstein

Atari User's Guide

Publishing Director: David Culverwell
Acquisitions Editor: Terrell Anderson
Production Editor/Text Designer: Michael J. Rogers
Art Director/Cover Design: Don Sellers
Assistant Art Director: Bernard Vervin
Photography: George Dodson
Manufacturing Director: John Komsa

Copy Editor: Carol Thorsten-Stein
Typesetter: Alexander Typesetting, Inc., Indianapolis, IN
Printer: Fairfield Graphics, Fairfield, PA
Typefaces: Souvenir (display), Palatino (text), OCR-B (computer programs and related material)

The ATASCII table found in Table 6-4 and Appendix D is from *Your Atari Computer* by Lon Poole with Martin McNiff and Steven Cook, © 1982 by McGraw-Hill, Inc. Used with permission of Osborne/McGraw-Hill.

Atari User's Guide

Mark C. Ellis
Dr. Robert Ellis
Dr. Larry Joel Goldstein

Brady Communications Company, Inc.
A Prentice-Hall Company
Bowie, MD

Atari User's Guide

Copyright © 1984 by Brady Communications Company, Inc.
All rights reserved. No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including, photocopying and recording, or by any information storage and retrieval system, without permission in writing from the publisher. For information, address Brady Communications Company, Inc., Bowie, MD 20715.

Library of Congress Cataloging in Publication Data

Ellis, Mark C., 1969
Atari user's guide.

Includes index.
1. Atari computer—Programming. 2 BASIC (Computer program language) I. Ellis, Robert, 1938- . II. Goldstein, Larry Joel. III. Title.
QA76.8.A82E44 1984 001.64'24 83-26625

ISBN 0-89303-323-5

Prentice-Hall International, Inc., London
Prentice-Hall Canada, Inc., Scarborough, Ontario
Prentice-Hall of Australia, Pty., Ltd., Sydney
Prentice-Hall of India Private Limited, New Delhi
Prentice-Hall of Japan, Inc., Tokyo
Prentice-Hall of Southeast Asia Pte. Ltd., Singapore
Whitehall Books, Limited, Petone, New Zealand
Editora Prentice-Hall Do Brasil LTDA., Rio de Janeiro

Printed in the United States of America

84 85 86 87 88 89 90 91 92 93 94 1 2 3 4 5 6 7 8 9 10

PREFACE

This book is designed to teach the computer novice how to use the ATARI family of computers, including the 400, 600XL, 800, 800XL, and 1200XL.

The development of the personal computer is one of the most exciting breakthroughs of our time. Indeed, the inexpensive personal computer promises to bring the computer revolution to millions of people and to alter the way they think, learn, work, and play. This book is intended to be an introduction to this revolution. Accordingly, the book has two main purposes. First, it instructs you in the operation and programming of ATARI computers and, secondly, it illustrates some of the many ways computers may be used. In addition, it acts as a handy reference guide, with useful information in the appendices.

We guide you from the moment you first turn on the computer: We discuss the rudiments of programming in BASIC; show you how to use paddles, joysticks, cassette recorders, disk drives, and printers; explain the sound and graphics capabilities of ATARI computers; and give guidelines for making simple games for the computer. There are numerous programs in the text to illustrate not only the programming techniques presented, but also the use of sound and graphics effects. We have included several applications that illustrate how the computer can be used in real-life situations. There are optional chapters on data files and the use of mathematical functions in computuing. We even provide suggestions for further exploration of the expanding computer world.

Since the book is designed for self-study, exercises of varying difficulty are provided. Located in the text of each section are boxed questions labeled "TEST YOUR UNDERSTANDING." These questions test your comprehension of concepts introduced in the section. The answers to "TEST YOUR UNDERSTANDING" questions are found after the exercises for each section.

Any book owes its existence to the dedicated labor and inspiration of many people. In our case, we have been inspired by our families and friends. Special thanks go to former teachers Carmela Mannarino and William Lopes. Our sincere thanks to our reviewers, Gerald Isaacs, Mike Dunn, Greg Leslie, and Mark Levine, for their careful scrutiny of the manuscript and many helpful suggestions. Thanks also go to Michael Rogers, production editor, for the professional manner in which he managed the editing and production of this book. Finally, we would like to thank Charlie Siegel, President of the Brady Company, and David Culverwell, Publishing Director of the Brady Company, for their continued support.

Mark C. Ellis
Dr. Robert Ellis
Dr. Larry Joel Goldstein
Silver Spring, Maryland

CONTENTS

Limits of Liability and Disclaimer of Warranty

The authors and publisher of this book have used their best efforts in preparing this book and the programs contained in it. These efforts include the development, research, and testing of the theories and programs to determine their effectiveness. The authors and publisher make no warranty of any kind, expressed or implied, with regard to these programs or the documentation contained in this book. The authors and publisher shall not be liable in any event for incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of these programs.

Note to Authors

Do you have a manuscript or a software program related to personal computers? Do you have an idea for developing such a project? If so, we would like to hear from you. The Brady Co. produces a complete range of books and applications software for the personal computer market. We invite you to write to David Culverwell, Publishing Director, Robert J. Brady Co., Bowie, MD 20715.

Trademarks of Material in This Text

ATARI 400, 600XL, 800, 800XL, and 1200XL are registered trademarks of Atari, Inc.
ATARIWRITER is a registered trademark of Atari, Inc.
ATARI 850 interface module is a registered trademark of Atari, Inc.

1

A First Look at Computers

1.1 Introduction

The computer age is barely thirty years old, but it has already had a profound effect on all our lives. Indeed, computers are now common in the office, the factory, and even the supermarket. In the last three or four years, the computer has even become common in the home, as people have purchased millions of computers and computer games. Computers are so common today that it is hard to imagine even a single day in which a computer will not somehow affect us.

In spite of the explosion of computer use in our society, most people know very little about them. They view a computer as an "electronic brain," and do not know how one works, how it may be used, or how greatly it may simplify various everyday tasks. This does not reflect a lack of interest. Most people realize computers are here to stay, and are interested in finding out how to use them. If you are one of those people, this book is for you.

This book is an introduction to personal computing for the novice. You may be a student, teacher, homemaker, business person, or just a curious individual. We assume you have had little or no previous exposure to computers and want to learn the fundamentals. We will guide you as you turn on your ATARI personal computer for the first time. From there, we will lead you through the fundamentals of communicating with your computer in a language called ATARI BASIC. Throughout, we will provide exercises for you to test your understanding of the material. We will show the many ways you can use your computer. In the exercises, we will suggest programs to write. Many of the exercises will be designed to give you insight into how computers are used in business and industry. We will suggest a number of applications of the computer within your home. For good measure, we'll even provide a few computer games!

What is Personal Computing?

In the early days of computing (the 1940s and 1950s), the typical computer was a huge mass of electronic parts which occupied sev-

eral rooms. In those days, it was often necessary to reinforce the floor of a computer room, and install special air conditioning so the computer could function properly. Moreover, an early computer was likely to cost several million dollars.

Over the years, the cost of computers has decreased dramatically and, thanks to micro-miniaturization, their size has shrunk even faster than their price.

In the late 1970s, the first "personal" computers were put on the market. These computers were reasonably inexpensive and were designed to allow the average person to learn about the computer and use it to solve everyday problems. These personal computers proved to be incredibly popular and have stirred the imaginations of people in all walks of life. It is no exaggeration to say that a computer revolution is now under way, as millions of people are learning to fit computers into their everyday lives.

The personal computer is not a toy, although it can be used as one. It is a genuine computer that can be equipped with enough capacity to handle the accounting and inventory control tasks of most small businesses. It can also perform computations for engineers and scientists, and can even be used to keep track of home finances and personal clerical chores. It would be quite impossible to give a complete list of all the applications of personal computers. However, the following list is suggestive of the range of possibilities:

For the business person

- Accounting
- Record keeping
- Clerical chores
- Inventory
- Cash management
- Payroll
- Graph and chart preparation
- Word processing
- Data analysis

For the home

- Record keeping
- Budget management
- Investment analysis
- Tax preparation
- Correspondence
- Energy conservation
- Home security

For the student

Computer literacy
Preparation of term papers
Analysis of experiments
Preparation of graphs and charts
Project schedules
Storage and organization of notes
Educational programs

For the professional

Billing
Analysis of data
Report generation
Correspondence

For recreation

Computer games
Computer graphics
Computer art
Computer music

As you can see, the list is quite comprehensive. If your interests aren't listed, don't worry! There's plenty of room for those of you who are just plain curious about computers and wish to learn about them as a hobby.

ATARI Personal Computers

This book will introduce you to personal computing on the ATARI 400, 600XL, 800, 800XL, and 1200XL personal computers. These machines are incredibly sophisticated devices which incorporate many of the features of their main-frame big brothers. Before we begin to discuss the particular features of the ATARI personal computers, let us begin by discussing the features found in all computers.

1.2 What is a Computer?

At the heart of every computer is a **central processing unit (CPU)** that performs the commands you specify. This unit carries out arithmetic, makes logical decisions, and so forth. In essence, the CPU is the "brain" of the computer. The memory of a computer enables it to "remember" numbers, words, and paragraphs,

as well as the list of commands you wish the computer to perform.

In ATARI computers, the CPU is contained in a tiny electronic chip, called a 6502 microprocessor. For a computer novice, it will not be necessary to know about the electronics of the CPU, so we will not discuss it further.

There are two types of memory inside computers: ROM and RAM. **ROM** stands for **Read Only Memory.** This type of memory contains information that is used by the CPU to operate the computer. The information is pre-recorded in the factory and can NEVER be erased. All computers have been designed in such a way that you can't store anything in ROM. This prevents interference with the operation of the computer.

RAM stands for **Random Access Memory.** This is the memory into which you can store information. When you type characters on the keyboard, they are stored in RAM. Similarly, results of calculations are kept in RAM awaiting output to you.

There is an extremely important fact about RAM which you should always remember:

When the computer is turned off, ALL information in RAM is erased.

RAM is used as the computer's main working storage area because of its quick accessibility. It takes about a millionth of a second to store or retrieve a piece of data from RAM. RAM may

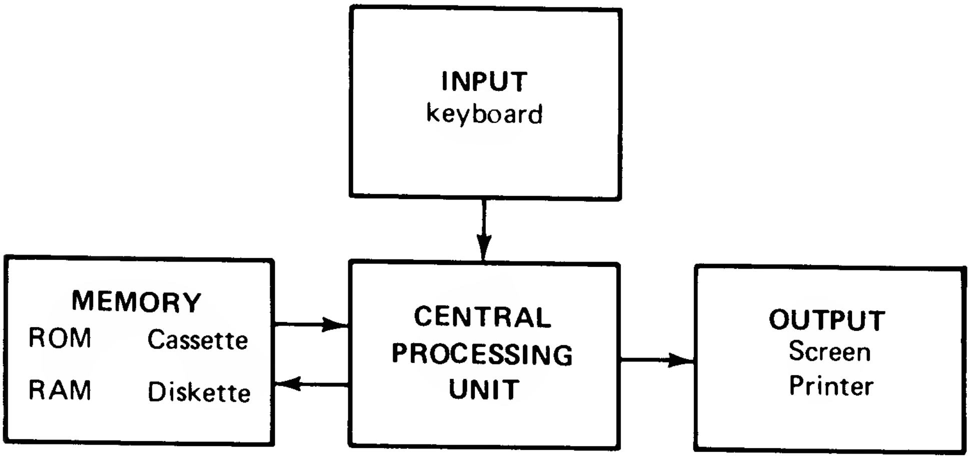

Figure 1-1. The main components of a computer.

not be used to store data in permanent form since it is erased when the computer is turned off.

To make permanent and retrievable copies of programs and data, we may use either cassettes or diskettes, which we will discuss later. An input device allows you to send information to the computer. Data sent to a computer is called **input.** An output device accepts data sent from the memory of the computer. Data sent by a computer is called **output.** The relationships of the four basic components of a computer are shown in Figure 1-1.

The main input device of an ATARI computer is the keyboard. We will discuss the special features of the keyboard in Sections 1.4 and 1.5. For now, think of the keyboard as a typewriter. By typing symbols on the keyboard, you are inputting them to the computer.

ATARI computers have a number of output devices. The most basic is the TV screen (sometimes called a video monitor, video display, or CRT). You may also use a printer to provide output on paper. In computer jargon, a printed output is called a **hard copy.**

Two of the devices that can be used for both input and output are the cassette recorder and the disk drive.

The cassette recorder is just a tape recorder which allows recording of information in a form which the computer can understand. The recording tape is the same sort you use for musical recordings. The main disadvantage of using a cassette recorder is its slowness.

The ATARI disk drive records information on flexible diskettes. The diskettes are often called "floppy disks," and can store about 92 thousand characters each! (A double-spaced typed page contains from 1,000 to 3,000 characters.) A disk drive can provide access to information in much less time than a cassette recorder, and is much more reliable. On the other hand, disk drives are more costly than cassette recorders.

We will go into further detail about cassette recorders and disk drives in Chapter 5.

1.3 Meet Your ATARI Computer

The best way to become quickly acquainted with your ATARI computer is to read this book while sitting at the computer. That way, you can verify the various statements as they come up and type in programs. So why not have a seat in front of your computer and become acquainted with it? We suggest you begin this

section by skipping ahead to the part that deals with your particular computer.

> **CAUTION:** NEVER use your computer during a storm unless you have a power surge protector. This protects your computer from surges and dips in electricity that may be caused by lightning. Make sure there isn't too much static electricity in the room you are in. Even a small spark can erase and destroy your computer's memory.

Figure 1-2. The ATARI 400 computer.

The ATARI 400

In case your computer is not conveniently available, we have provided pictures of it and its keyboard in Figures 1-2 and 1-3.

Notice in Figure 1-2 we placed the monitor behind the computer for convenient viewing. If you wish, you may place your monitor beside the computer. The choice is yours, so arrange your system for your comfort and convenience. You may be using your system for many hours at a time and a little convenience will lessen eye-strain and fatigue.

Now open the hatch at the top of the computer, insert your BASIC cartridge into the slot (see Figure 1-4), close the hatch, and turn on the computer. On the screen you will see

```
READY
■
```

This is called the **ready prompt.** It is the computer's way of telling you it is ready to receive instructions from you.

The keyboard of the ATARI 400 is similar to that of a typewriter, except it has a flat, pressure-sensitive surface. It has keys for A-Z, 0-9, and other symbols such as punctuation marks. Some special keys are ESC, CTRL, CLEAR, INSERT, DELETE BACK S, BREAK, CAPS LOWR ,THE ATARI KEY (⅄), START, SELECT, OPTION, and SYSTEM RESET. Don't panic! We will discuss the functions of these keys as we need them.

Figure 1-3. Keyboard of the ATARI 400 computer.

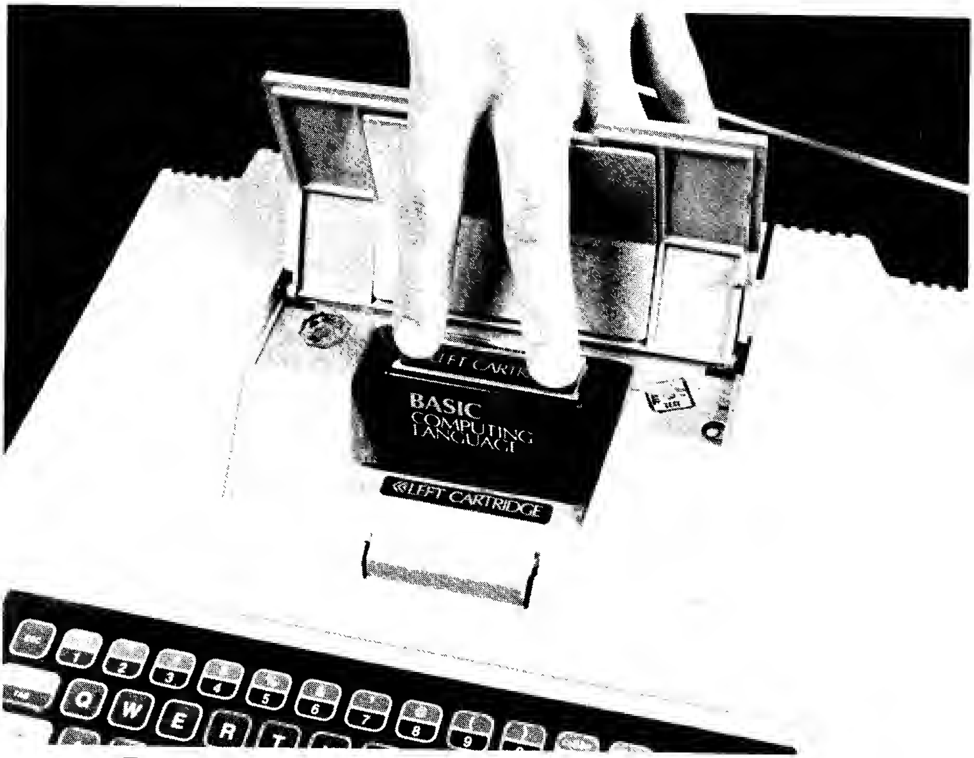

Figure 1-4. Inserting the BASIC cartridge in the ATARI 400.

The ATARI 600XL

In case your computer is not conveniently available, we have provided pictures of it and its keyboard in Figures 1-5 and 1-6.

Notice in Figure 1-5 we placed the monitor behind the computer for convenient viewing. If you wish, you may place your monitor beside the computer. The choice is yours, so by all means arrange your system for your comfort and convenience. You may be using your system for many hours at a time and a little convenience will lessen eyestrain and fatigue.

The slot just above the keyboard (see Figure 1-5) is used for inserting cartridges with games, languages, word processers, or other programs.

XL BASIC, a variation of ATARI BASIC, is built into the 600XL. Therefore, no BASIC cartridge is necessary. When you turn on the computer, you will see

```
READY
■
```

This is called the **ready prompt.** It is the computer's way of telling you it is ready to accept information.

The keyboard of the 600XL is similar to that of a typewriter. It has keys for A-Z, 0-9, and other symbols such as punctuation marks. Some special keys are RESET, START, SELECT, OPTION,

Figure 1-5. The ATARI 600XL computer.

Figure 1-6. Keyboard of the ATARI 600XL computer.

HELP, THE ATARI KEY (◪), BREAK, ESC, CONTROL, CLEAR, INSERT, DELETE BACK SPACE, and CAPS. Don't panic! We will discuss the functions of these keys as we need them. In the rest of this book, we will abbreviate CONTROL to CTRL and DELETE BACK SPACE to DELETE BACK S when speaking generally about the ATARI computers. We will also refer to the CAPS key as CAPS LOWR and RESET as SYSTEM RESET.

The ATARI 800

In case your computer is not conveniently available, we have provided pictures of it and its keyboard in Figures 1-7 and 1-8.

Figure 1-7. The ATARI 800 computer.

Notice in Figure 1-7 we placed the monitor behind the computer for convenient viewing. If you wish, you may place your monitor beside the computer. The choice is yours, so arrange your system for your comfort and convenience. You may be using your system for many hours at a time and a little convenience will lessen eye-strain and fatigue.

Open the hatch at the top of the computer, insert your BASIC cartridge into the left slot (see Figure 1-9), close the hatch, and turn on the computer. On the screen you will see

```
READY
■
```

This is called the **ready prompt.** It is the computer's way of telling you it is ready to receive instructions from you.

The keyboard of the 800 is similar to that of a typewriter. It has keys for A-Z, 0-9, and other symbols such as punctuation marks and mathematical symbols. Some special keys are ESC, CTRL, CLEAR, INSERT, DELETE BACK S, BREAK, CAPS LOWR, THE ATARI KEY (∧), START, SELECT, OPTION, and SYSTEM

Figure 1-8. Keyboard of the ATARI 800 computer.

Figure 1-9. Inserting the BASIC cartridge in the ATARI 800.

RESET. We will discuss the functions of these keys as we need them.

The ATARI 800XL

In case your computer is not conveniently available, we have provided pictures of it and its keyboard in Figures 1-10 and 1-11.

Notice in Figure 1-10 we placed the monitor behind the computer for convenient viewing. If you wish, you may place your monitor beside the computer. The choice is yours, so by all means arrange your system for your comfort and convenience. You may be using your system for many hours at a time and a little convenience will lessen eyestrain and fatigue.

The slot just above the keyboard (see Figure 1-10) is used for inserting cartridges with games, languages, word processers, or other programs.

Figure 1-10. The ATARI 800XL computer.

Figure 1-11. Keyboard of the ATARI 800XL computer.

XL BASIC, a variation of ATARI BASIC, is built into the 800XL. Therefore, no BASIC cartridge is necessary. When you turn on the computer, you will see

```
READY
■
```

This is called the **ready prompt.** It is the computer's way of telling you it is ready to accept information.

The keyboard of the 800XL is similar to that of a typewriter. It has keys for A-Z, 0-9, and other symbols such as punctuation marks. Some special keys are RESET, START, SELECT, OPTION, HELP, THE ATARI KEY (◪), BREAK, ESC, CONTROL, CLEAR, INSERT, DELETE BACK SPACE, and CAPS. Don't panic! We will discuss the functions of these keys as we need them. In the rest of this book, we will abbreviate CONTROL to CTRL and DELETE BACK SPACE to DELETE BACK S when speaking generally about the ATARI computers. We will also refer to the CAPS key as CAPS LOWR and RESET as SYSTEM RESET.

The ATARI 1200XL

In case your computer is not conveniently available, we have provided pictures of it and its keyboard in Figures 1-12 and 1-13.

Notice in Figure 1-12 we placed the monitor behind the computer for convenient viewing. If you wish, you may place your monitor beside the computer. The choice is yours, so by all means arrange your system for your comfort and convenience. You may be using your system for many hours at a time and a little convenience will lessen eyestrain and fatigue.

Insert the BASIC cartridge into the left side of your computer (see Figure 1-14) and then turn the computer on.

On the screen you will see

```
READY
■
```

This is called the **ready prompt.** It is the computer's way of telling you it is ready to accept information. (If you insert the BASIC cartridge after turning on the computer, you may see nothing on the screen, or you may see colors moving on the screen. If so, just turn the computer off and then back on.)

The keyboard of the 1200XL is similar to that of a typewriter. It has keys for A-Z, 0-9, and other symbols such as punctuation marks. Some special keys are RESET, START, SELECT, OPTION, F1, F2, F3, F4, HELP, THE ATARI KEY (◪), BREAK, ESC, CONTROL, CLEAR, INSERT, DELETE BACK SPACE, and CAPS. Don't panic! We will discuss the functions of these keys as we need them. In the rest of this book, we will abbreviate CONTROL to CTRL and

DELETE BACK SPACE to DELETE BACK S when speaking generally about the ATARI computers. We will also refer to the CAPS key as CAPS LOWR and RESET as SYSTEM RESET.

Figure 1-12. The ATARI 1200XL computer.

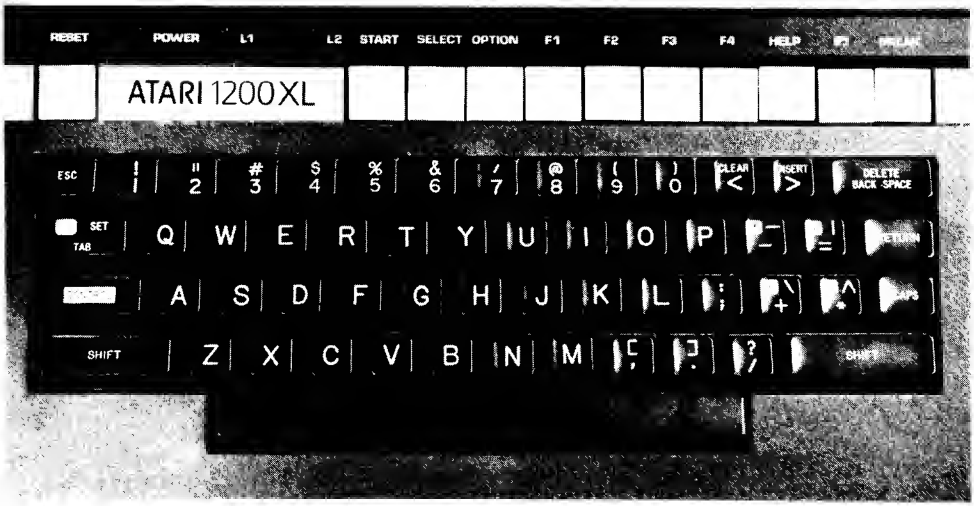

Figure 1-13. Keyboard of the ATARI 1200XL computer.

Figure 1-14. Inserting the BASIC cartridge into the ATARI 1200XL.

1.4 Using the Keyboard

In this section, you will learn how to use some of the keys on your computer.

Try typing "HELLO" on the keyboard. Notice the word "HELLO" appeared on the screen in capital letters, and the computer (or TV if you are using a 600XL, 800XL, or 1200XL) made a clicking sound each time you pressed a key. Now press the RETURN key. On the screen you will see

```
ERROR- HELLO■
```

Don't worry about this! You didn't do anything wrong. The reason for the error is that the computer does not understand what you typed.

Whenever you hit the RETURN key, the computer tries to interpret what you have typed, but it understands only its own language, in this case ATARI BASIC. Whenever the computer encounters something it does not understand, it conveniently gives you an error message to let you know. Later with this book, you

will learn ATARI BASIC so you can make yourself understood to the computer. For now, just ignore error messages.

Before we proceed, let us make an important point. You cannot damage your computer by pressing any of the keys in a normal fashion. The worst that can happen is that you will erase something from the screen that you wanted to keep. In any case, you can always turn the computer off and then back on. So please don't worry about damaging the computer.

The computer always prints with capital letters unless you press the CAPS LOWR key. Try pressing this key and then typing "HELLO" again. This time the computer put "hello" on the screen in lower case letters. To get the computer back to upper case, press SHIFT and CAPS LOWR simultaneously*. Try experimenting until you fully understand the CAPS LOWR key.

As with a typewriter, you always use the SHIFT key in conjunction with other keys, not by itself. The same applies to the CTRL key. You could consider it a second SHIFT key, with its own separate purpose.

Notice that the numbers 0-9 are at the top of the keyboard. It is not necessary to use the upper case O (as in Ontario) for the number zero, or the lower case l (as in large) for the number one. Unless the computer is given instructions to the contrary, it will always interpret the lower case l and upper case O as letters, not numbers.

Several of the special keys make it easy to change whatever you or the computer has placed on the screen. Making such changes is called **screen editing.** For example, to clear the screen all you have to do is to press SHIFT and CLEAR (or CTRL and CLEAR) simultaneously. Try it! It is frequently convenient to clear the screen for added clarity.

By now you have probably noticed a little white square on the screen. It is called the **cursor.** The cursor's "home" is at the top left corner of the screen. That is where it is located right after the screen has been cleared. Its location marks the place where the next character you type will be put on the screen. It advances automatically each time you type a character (this includes the space character). At the end of a line, the cursor will advance to the beginning of the next line. (In particular, this means that it is not necessary to hit RETURN at the end of a line to go to the next line. The computer does it for you!)

---

*When we ask you to press 2 keys simultaneously, press and hold the first key while pressing the second key.

You may change the position of the cursor by using CTRL with any of the four keys on the right side of the keyboard that contain arrows. (On the 1200XL, F1 through F4 can also be used to move the cursor.) To move the cursor, simultaneously press CTRL and the key with the arrow that points in the direction you want the cursor to move. Pressing the keys briefly will advance the cursor one space in the chosen direction; holding the keys down will cause the cursor to continue moving until you release the key with the arrow. If the cursor disappears from either side of the screen, it will reappear on the other side of the screen. The same sort of thing happens when the cursor disappears from the top or bottom of the screen. Practice moving the cursor until you understand the procedure fully.

TEST YOUR UNDERSTANDING 1 (answer on page 19)

Move the cursor to near the center of the screen, type your name, and clear the screen. Where does the cursor move when the screen is cleared? What is the name of this place?

Next we explain the use of the keys INSERT and DELETE BACK S. Type the following exactly:

```
USING THE KOMPUTR IS FUN.....
```

There are several errors in this sentence. The letter K should be C, there is an E missing between T and R, and there are too many periods at the end of the sentence.

Your cursor should now be to the right of the last period. If it is not, please place it there at this time.

To delete all the periods but one, press DELETE BACK S four times. Each time you press this key, the cursor will move one space to the left, deleting the character in the space to which it moves. This is what should be displayed on the screen at this time:

```
USING THE KOMPUTR IS FUN.
```

Now let's correct the word KOMPUTR. What a mess! First you must insert the letter E between T and R. Move the cursor until it is on top of the R. Then press CTRL and INSERT simultaneously. This will shove everything from the cursor on one space to the right, creating a space between the T and R. Now you can type an E, leaving the following displayed on the screen:

```
USING THE KOMPUTER IS FUN.
```

Finally, you must change the K to a C. Just position the cursor on top of the letter K and type the letter C. This will replace the K with a C. Your sentence now reads

```
USING THE COMPUTER IS FUN.
```

There is another use of DELETE BACK S that is frequently useful. Suppose you wanted to delete the word 'USING' from your sentence. Move the cursor until it is located on top of the 'U' in 'USING'. While holding the CTRL key down, press DELETE BACK S five times. Each time you do so, the space occupied by the cursor is deleted and everything after the cursor is retracted one space.

If you now press and hold both CTRL and DELETE BACK S, the cursor will appear to "eat up" the words THE COMPUTER IS FUN. If you had pressed SHIFT with DELETE BACK S, an entire line would have been deleted. Similarly, if you had pressed SHIFT with INSERT, an entire line of empty space would have been created, moving the existing line down.

TEST YOUR UNDERSTANDING 2 (answer on page 19)

Type the following exactly as it is and then make all the necessary corrections:

Computers aret geting beter and better all the time. They have more membory and are getting smaler andd faster. Just a few years ago,

e,o,got,ho,ogpeewrp;

computers were mush more costly, too.

The special keys we have not discussed will be explained later.

ANSWERS TO TEST YOUR UNDERSTANDINGS 1 and 2

1: Upper lefthand corner of screen; home

2: Computers are getting better and better all the time. They have more memory and are getting smaller and faster. Just a few years ago, computers were much more costly, too.

1.5 Special Keys for the ATARI 600XL, 800XL, and 1200XL

Many keys are unique to the ATARI XL computers. One of these is the HELP key. With this key, you can test various parts of the computer. There is a **memory test,** an **audio-visual test,** and a **keyboard test**

To use these tests on the 600XL or 800XL, turn the computer on with no cartridge inserted while holding down the OPTION key. After a few seconds, you will see a large selection menu. Press SELECT until you get to the test you want. Then press the START key. The test you selected will commence.

To use these tests on the 1200XL, turn on your computer with no cartridge inserted. After a few seconds you will see the word ATARI in large, multi-colored letters. At this point, press the HELP key. You will see a selection menu. Press SELECT until you get to the test you want. Then press the START key. The test you selected will commence.

Each test is repeated indefinitely until you press RESET or HELP. With the 600XL and 800XL, RESET takes you back to BASIC. With the 1200XL, RESET takes you back to the word ATARI. On all XL computers, HELP takes you back to the selection menu.

The **memory test** is a test to check all parts of memory. After a few seconds, you should see 2 long rectangular green strips. Under them you should see small green boxes begin to appear one at a time. The long rectangles are for testing ROM, and the small squares for RAM. If any of the ROM rectangles or RAM squares are red, you should see a qualified ATARI dealer to have your machine fixed.

The **audio-visual test** checks your computer's color and sound capibilities. During the test, you will see a treble clef displayed, with the voice number of the voice being tested under it. Six notes are displayed and played sequentially on the screen. If any of these notes does not play or is off key, you should get your computer fixed.

The **keyboard test** will display a keyboard on the screen. If you press a key, the corresponding key on the screen will flash. SHIFT and CONTROL will only flash if another key is pressed with them. BREAK, HELP, and RESET do not flash when you press them. If the test is conducted on any computer other than an ATARI 1200XL, there are no keys labeled F1, F2, F3, or F4. Therefore, the corresponding keys on the simulated keyboard should be ignored.

The **all tests** option will go through all the tests one at a time. With the **all tests** option, when the computer gets to the keyboard test, it automatically tests the keyboard by pressing random keys.

The ATARI 1200XL has four other keys that are not available on any of the other ATARI computers. These are the F1, F2, F3, and F4 keys. Used by themselves, they move the cursor. F1 moves the cursor up, F2 down, F3 to the left, and F4 to the right.

These keys can also be used in conjunction with SHIFT or CONTROL. When used with CONTROL, they can deactivate the computer's keyboard (so you can leave the computer on without worrying that someone will interfere with your program), turn off the output to the screen (so the computer can work faster), turn off the key-click output to the TV speaker, or switch the keyboard to a set of international characters which can be printed by pressing CONTROL in conjunction with any lettered key (see Figure 1-15). Table 1-1 summarizes all the uses of these keys.

Table 1-1. The uses of the F1, F2, F3, and F4 keys available on the 1200XL.

KEYSTROKES	*EFFECT*
F1	Moves cursor up.
F2	Moves cursor down.
F3	Moves cursor to left.
F4	Moves cursor to right.
SHIFT & F1	Moves cursor to upper left of screen.
SHIFT & F2	Moves cursor to lower right of screen.
SHIFT & F3	Moves cursor to beginning of line.
SHIFT & F4	Moves cursor to end of line.
CONTROL & F1	Deactivates keyboard. Press CONTROL & F1 again to reactivate.
CONTROL & F2	Turns off output to screen. Press any key except SHIFT, BREAK, CONTROL, OPTION, START, or SELECT to turn back on.
CONTROL & F3	Turns off key-click. Press CONTROL and F3 to turn back on.
CONTROL & F4	Switches keyboard to international character set. See Figure 1-15. Press CONTROL and F4 to return keyboard.to normal

Table 1-1. The uses of the F1, F2, F3, and F4 keys available on the 1200XL (continued).

To switch to the international character set on the 600XL and 800XL, type

```
POKE 756,204
```

To switch the character set back to normal, type

```
POKE 756,224
```

Figure 1-15. The international character set on the ATARI computers using CONTROL.

2

Getting Started in ATARI BASIC

2.1 Computer Languages

In the last chapter we learned how to manipulate the keyboard of ATARI computers and how to do screen editing. In this chapter, we'll learn how to give instructions to the computer.

Just as humans use languages to communicate with one another, computers use languages to communicate with other electronic devices (such as printers), human operators, and even other computers. There are hundreds of computer languages in use today. Your ATARI computer is capable of "speaking" quite a few of them, including ATARI BASIC, which is both versatile and easy to learn.

ATARI BASIC is a version of BASIC, a language developed especially for computer novices by John Kemeny and Thomas Kurtz at Dartmouth College. Most home computers have their own version of BASIC.

In the next few chapters, we will concentrate on learning the fundamentals of ATARI BASIC. In the process, we will learn a great deal about the way in which a computer may be used to solve problems.

In order to use ATARI BASIC, you must have a BASIC cartridge in the computer and have the computer on unless you have the 600XL or 800XL (see Section 1.3).

2.2 Printing With the Direct Mode

In this section we will be using the **direct** (or **immediate**) **mode.** This means you can give the computer certain instructions and it will execute each instruction immediately after you hit RETURN. An instruction to the computer is called a **statement** or **command.** You will eventually learn lots of statements and commands, but for now, let's begin with a few simple ones.

The PRINT Statement

The PRINT statement tells the computer to print something on the screen. Type the following exactly and then hit RETURN:

```
PRINT "DON'T FORGET THE QUOTATION MARKS"
```

Immediately after you hit RETURN, the computer should have printed

```
DON'T FORGET THE QUOTATION MARKS
```

on the screen directly under the the line with the PRINT statement. If not, maybe you forgot to type in the quotation marks.

Actually, you can omit the second quotation mark and the computer will still understand you. For example, if you type

```
PRINT "HELLO
```

the computer will print

```
HELLO
```

on the screen when you hit RETURN. But if you type

```
PRINT HELLO
```

or

```
PRINT HELLO"
```

the computer will not give you the desired result.

There is an exception to the rule of needing quotation marks: They are not needed for printing numbers. To see this, try typing

```
PRINT 1
```

The computer will print the number 1 on the screen.

> **IMPORTANT:** You may have noticed the computer did not print any of your words until you hit RETURN. Hitting RETURN is your way of telling the computer you want it to execute the command that you have typed in. This is true of the PRINT statement and all the other statements and commands we will discuss later on. Usually we will not remind you to hit RETURN after typing a statement. We expect you to do it automatically.

The statement PRINT can be abbreviated to either PR. or ?. Thus the following three statements have exactly the same meaning to the computer:

```
PRINT "HELLO"
PR. "HELLO"
? "HELLO"
```

It is possible to give the computer more than one command in the direct mode before hitting RETURN. It is necessary to separate these commands with colons so the computer knows where one command ends and the next begins. When you hit RETURN, the computer will execute all of the commands (as long as they do not occupy more than three lines on the screen). For example, try typing

```
PRINT "COPY":PRINT "CAT"
```

When you hit RETURN, the computer will print

```
COPY
CAT
```

Notice that when the computer executed the second PRINT statement, it skipped to the next line on the screen and printed "CAT" beneath "COPY". If we wanted the two words next to each other on the same line, we could let the computer know by typing a semicolon between the two words and using a single PRINT statement

```
PRINT "COPY";"CAT"
```

The output on the screen would be

```
COPYCAT
```

The same result could also be obtained via the instructions

```
PRINT "COPY";:PRINT"CAT"
```

Of course, we could have achieved the same result with no semicolon and a single set of quotation marks, but you will see later how useful the semicolon can be.

If we wanted the two words on the same line but with a space between them, we would include the space in one of the sets of quotation marks. We could type either

```
PRINT "COPY ";"CAT"
```

or

```
PRINT "COPY";" CAT"
```

and the computer would print

```
COPY CAT
```

If we wanted several spaces between the words, we could include as many spaces in the quotation marks as we wanted, or

we could use a comma in place of the semicolon. To see the effect of the comma, type the following with twelve ones:

```
PRINT 1,1,1,1,1,1,1,1,1,1,1,1
```

The output will be

```
1         1         1         1
  1         1         1         1
    1         1         1         1
```

The PRINT statement works in a special way with a comma. Each time the computer encounters a comma (not inside a set of quotation marks) in a PRINT statement, it next begins printing at the start of the next printing zone. Each printing zone is 10 spaces wide.

The ones in the above display indicate the starting points of the first twelve printing zones. Notice that the first one on the second line is indented two spaces. This is due to the fact that the computer prints in 38 spaces on each horizontal line on the screen. (In Section 3.6 we will learn how to alter the length of these lines with the POKE statement.) This means that only eight spaces of the fourth printing zone are on the first line. The other two are on the second line. Therefore, the fifth printing zone must begin with the third space on the second line rather than the first. Type in the following example, which illustrates one use of the comma:

```
PRINT "PRICE","TAX","TOTAL":PRINT
"$18.95","$0.95","$19.90"
```

The output on the screen will be

```
PRICE     TAX       TOTAL
$18.95    $0.95     $19.90
```

So far, we have used the comma only to print in the next printing zone. It is also possible to skip printing zones. This is done simply by using two or more commas (instead of one) with a PRINT statement. Here are two examples of this:

```
PRINT "HELLO",,"GOOD-BYE"
PRINT ,,"HELLO"
```

The first example will print HELLO on the screen, skip two printing zones, and then print GOOD-BYE. The second example will skip 2 printing zones and then print HELLO.

TEST YOUR UNDERSTANDING 1 (answers on page 31)

Have the computer print out each of the following expressions. Use a colon, semicolon, or comma, if possible.

a.
```
BACK
SPACE
```
b.
```
BACKSPACE
```
c.
```
BACK SPACE
```
d.
```
BACK          SPACE
```
e.
```
BOOKS         FOOD          TOTAL
$132          $1050         $1182
```
f.
```
BOOKS                       FOOD
$132                        $1050
```

Strings

From the preceding examples, you can see it is possible to print on the screen any sequence of keyboard characters. An exception is the quotation mark(") since it has special significance for the PRINT statement. A sequence of keyboard characters is called a **string** or a **string constant.** When we refer to strings, we will usually enclose them in quotation marks. Here are a few examples:

```
"HELLO", "A1b2", "345", "A Z"
```

A space is considered a keyboard character. For example, the string "A Z" is different from the string "AZ". The computer also treats the string "345" differently than it does the number 345. For example, it will not add the strings "345" and "846".

Arithmetic Operations

Your computer can perform arithmetic operations with lightning speed. You can use the PRINT statement to instruct the computer to print out the results it obtains. Try typing this instruction:

```
PRINT 45+20
```

When you hit RETURN, the computer will print out the result, 65. Notice we did not enclose 45 + 20 in quotation marks. If we had, the computer would have printed the string "45 + 20". Notice also that the symbol for addition is the usual plus sign: +. The symbol for subtraction is the minus sign -, which is located on the key-

board on the same key as the arrow that points upward. It is not necessary to press SHIFT to type -. For an example of subtraction, type the statement

```
PRINT 45-20
```

The computer will print the result, 25, on the screen.

It is frequently convenient to combine strings and numbers in the same PRINT statement. Here is a nice example:

```
PRINT "40+25=";40+25
```

This will cause the computer to print

```
40+25=65
```

on the screen. This tells us the problem the computer solved as well as the result. It is good technique to have the computer print out information that will help you or someone else to understand what the computer has done.

The symbols for multiplication and division are * and /, respectively. For example, if you type

```
PRINT 45*20
```

the computer will print the product, 900, and if you type

```
PRINT 45/20
```

it will print the quotient, 2.25.

For many applications, you will need only the four arithmetic operations just discussed. However, it is sometimes necessary to raise a number to a power. For example, 3 to the power 5 means the product of five threes (3*3*3*3*3), which happens to equal 243. For the computer, this is typed as 3^5. The symbol ^ is on the same key as * and is typed by pressing that key simultaneously with SHIFT. In 3^5, the number 5 is the POWER or EXPONENT. For that reason, raising a number to a power is called **exponentiation.** Now try typing

```
PRINT 3^5
```

The 600XL and 800XL will print 243, the correct answer, on the screen. But the 400, 800, or 1200XL will print

```
242.999997
```

on the screen. This is not the exact answer, but it is very close. You just have to get used to the fact that the computer sometimes gives only approximate results. In fact, it always truncates (cuts off) a number to either 9 or 10 digits. To see how this works, type the following two statements:

```
PRINT 2.53148569876
```

and

```
PRINT 0.53148569876
```

The output on the screen will be

```
2.53148569
```

and

```
0.531485698
```

The computer handles very large and very small numbers in a different way. To see how it handles such numbers, type in the following two instructions:

```
PRINT 123456000000
```

and

```
PRINT 0.0000009876
```

The computer will print

```
1.23456E+11
```

and

```
9.876E-07
```

The E+11 means the decimal point in 1.23456 must be moved eleven places to the right to obtain the intended number, 123456000000. The E-07 means the decimal point in 9.876 must be moved seven places to the left to obtain the intended number, 0.0000009876.

In general, E with a plus sign means move the decimal to the right, and E with a minus sign means move the decimal point to the left. This type of notation is called **scientific notation.** Although you may not need to use this notation, at least you will know what it means if the computer uses it.

Notice we did not use any commas when we typed in the number 123456000000. This is because commas have a special significance to the computer, as we explained earlier. Thus the number 125,000 must be typed to the computer as 125000.

Order of Arithmetic Operations

Suppose you type

```
PRINT 2*3+4
```

What do you expect the computer to print? It must perform a multiplication and an addition, but in what order? If it first multiplies 2 times 3, getting 6, then adds 4, the final result will be 10. But if it first adds 3 and 4, getting 7, then muliplies by 2, the final result will be 14. As you can see, the final result depends on the order in which the operations are performed. In this case the computer will multiply first, so it will print 10 on the screen. If you wanted the computer to add 3 and 4 before multiplying by 2, you could use parentheses to let the computer know:

```
PRINT 2*(3+4)
```

The computer always performs the operations inside parentheses before those outside. Parentheses can be NESTED, that is, one set can be inside another. Here is an example:

```
PRINT (2*(3+4))^2
```

In this case, the computer would start inside the innermost parentheses and perform the operations in the following order:

1. Add 3 and 4, getting 7.
2. Multiply 2 and 7, getting 14.
3. Raise 14 to the power 2, getting 196

Here are some rules the computer follows:

1. It starts with operations inside innermost parentheses.
2. Within parentheses, it first performs exponentiation (if any), then multiplication and division (if any), then addition and subtraction (if any). The operations in each of these three groups are performed from left to right.

To see how the left to right feature works, type

```
PRINT 6/2*3
```

The computer will work from left to right, first dividing 6 by 2, resulting in 3, then multiplying 3 by 3 to get the final result, 9.

If you are ever in doubt as to what order the computer will follow, you can use extra sets of parentheses to get what you want. The computer won't mind at all. It never does.

Exercises (answers on page 247)

1. Have the computer perform the following operations and print the results on the screen:

   a. `31+62`
   b. `456-278`

c. `(45-23)/3`
d. `5*(45-(658+968))`

2. Have the computer perform each of the following operations and print the problem along with the result:

   a. `567+527`
   b. `23*67`
   c. `17^2`
   d. `(34-19)*23`

3. In each of the following, fill in the circles with the numbers 1, 2, 3,... to indicate the order in which the arithmetic operations would be performed by the computer:

   a. `56-○23+○45`
   b. `4+○5*○6`
   c. `(4+○5)*○6`
   d. `(24/○3*○4)^○(4+○7-○6)*○2`
   e. `((5-○3)*○17+○6)/○2+○6`

4. In each of the following examples, determine what the computer would print:

   a. `PRINT 9.58436589287`
   b. `PRINT 0.3571527948649`

5. What number is meant by each of the following:

   a. `4.69E+09`
   b. `1.234E-10`

6. How would the computer express each of the following in scientific notation:

   a. `245367000`
   b. `0.00000000000572`

ANSWERS TO TEST YOUR UNDERSTANDING 1

1: a. `PRINT "BACK":PRINT "SPACE"`
   b. `PRINT "BACK";"SPACE"`
   c. `PRINT "BACK ";"SPACE"`
   d. `PRINT "BACK","SPACE"`
   e. `PRINT "BOOKS","FOOD","TOTAL": PRINT "$132","$1050","$1182"`
   f. `PRINT "BOOKS",,"FOOD": PRINT "$132",,"$1050"`

2.3 Printing With the Program Mode

In the preceding section you learned how to give commands in the direct mode. There are several disadvantages to using the direct mode:

1. The commands are executed each time you press RETURN. Since you can type commands on only three lines before having to press RETURN, it is impossible to have the computer execute a long list of commands in the direct mode.
2. In the direct mode there is no easy way to preserve a list of commands for future use (tomorrow, perhaps).

It is possible to eliminate these difficulties by using the **program mode.** A **program** is a list of instructions to the computer. The basic idea of the program mode is that none of the instructions in a program are executed until all the instructions have been typed in and the computer has been instructed to begin.

To inform the computer that the instructions are to be executed in the program mode, you must number the lines in your program. After each numbered line, you must hit RETURN to let the computer know you have finished typing the line. It will automatically reposition the cursor at the start of the next line. Here is an example of a short program:

```
10 PRINT "PRICE"
20 PRINT "TAX"
30 PRINT "TOTAL COST"
40 END
```

If you type this into the computer, nothing will happen—yet. In order to instruct the computer to execute, or "run," the program, you must type

```
RUN
```

in the direct mode, that is, without a number in front of it. It is not necessary to have the program displayed on the screen when you run it. The computer stores the program in memory. When you run the preceding program, the computer will display

```
PRICE
TAX
TOTAL COST

READY
■
```

When you run a program, the computer executes the commands in the program according to the line numbers, beginning with the smallest number and proceeding from any number to the next higher number, unless instructed by the program to do otherwise. Don't worry about this possibility now. We'll explain it in Section 3.1.

In the program you just ran, the computer executed the lines in the order 10, 20, 30, 40. Lines 10, 20, and 30 instructed the computer to print something on the screen. Line 40 instructed the computer to stop and return to the direct mode.

It is good practice for the last command of a program to be END, although it is not necessary. When the computer reaches the highest numbered command, it will automatically stop, unless instructed to do otherwise.

It is also common practice for the line numbers to begin with 10 and to increase by 10s. However, you may use any line numbers you wish, as long as they are whole numbers from 0 to 32767. It is not necessary to type the lines in increasing order. The computer will automatically order them. If, for example, you type in lines 10, 20, 30, and 40, and then decide you need a line between lines 20 and 30, you could type in the line as line 25. This is an advantage of numbering the lines by 10's; you have intermediate numbers to use for new lines. We will almost always be numbering by 10's. We strongly suggest you do the same!

Before we proceed, we need to distinguish between two types of lines. A **display line** refers to the collection of spaces on a horizontal line on the screen. On ATARI computers, each display line normally contains 38 spaces. That number can be increased to 39 or 40 by changing the left margin (see Section 3.6.) A **program line** refers to a numbered line of a program. A program line may contain several statements separated by colons, but the maximum length of a program line is three display lines (normally 114 spaces.)

Running a program does not erase it from memory. To verify that your last program is still in memory, you could run it again by typing RUN, or you could type **LIST.** This command instructs the computer to print the program on the screen. The LIST command can be abbreviated to L. Thus the following two statements do the same thing:

```
LIST
L.
```

The LIST command will display the entire program in memory on the screen. This can be a disadvantage with long programs. You

may want to list only a part of the program. This is easily done. Just type LIST followed by 2 numbers separated by a comma. The numbers refer to the beginning and ending line numbers of the portion of the program to be listed. The second number must always be larger than the first. For example, the following command lists lines 17 through 28:

```
LIST 17,28
```

To list from any line number (57, for example) to the end of a program, type

```
LIST 57,32767
```

If you wish to erase a program from RAM, you may turn off the computer or you may type NEW. When you type NEW and hit RETURN, the computer erases the old program from memory and displays the READY prompt on the screen. The computer will not clear the screen after you type NEW. This means your old program might be on the screen, but not in memory! After you type NEW, you can enter another program into the computer.

> IMPORTANT:
> 1. Only one program can be in RAM at a time.
> 2. When the computer is turned off, any program in RAM will be erased.

Program Editing

There are several ways of editing (changing) a program in RAM:

1. To add a line to the program, just type the new line, being sure to give it a line number that will put the line where you want it in the program. It is not necessary to have the program displayed on the screen to add a line. For example, if you wanted the last program to print NUMBER OF ITEMS before PRICE, you could have typed in

   ```
   5 PRINT "NUMBER OF ITEMS"
   ```

   Type this in and then list the program to see that the computer accepted the new line and put it in the right place. If the computer did not accept your change, maybe you forgot to hit RETURN after typing your new line. Try running the program.
2. To delete a line of the program, type the line number and hit RETURN. Try this out on the last program: Type 5 and

hit RETURN. Then list the program. Line 5 should not appear. Try running the program.

3. To edit an existing line in a program, you have two options. You may simply retype the line as you want it, being sure to use the same line number it had before. When you do this and hit RETURN, the computer will erase the old line and replace it with the new line. Or you may list the program on the screen (if it is not already there) and use the screen editing techniques you learned in Section 1.4 to change the line as you wish. This method will normally be quicker if you need to make only minor corrections. Try both options on the program you have in memory.

IMPORTANT: Be sure to hit RETURN after changing any line. If you fail to do this, the computer will ignore the changes.

TEST YOUR UNDERSTANDING 1

Type the following program:

```
10 PRINT "HELLO"
20 PRINT "GOODBYE"
30 END
```

Run the program and have the computer list it on the screen. Then change the program so that it prints out "GOODBYE" first and then "HELLO". Then erase the program from memory without turning off the computer. Verify that the program has been erased by typing LIST and seeing how the computer responds.

2.4 Giving Names to Numbers and Words

As you start making larger programs, you will probably find yourself retyping certain numbers over and over. Not only does this retyping waste time and computer memory, it also is a likely source for errors. Fortunately, such retyping is unnecessary if we use **variables.**

A variable is a letter which is used to represent a number. Any letter of the alphabet may be used as a variable. There are other possible names for variables as we will explain later.

At any given moment, a variable has a particular value. For example, the variable A might have the value 5, while B might have the value -2.137845. One method for changing the value of a variable is through the use of the LET statement. The statement

```
LET A=7
```

sets the value of A equal to 7. Any previous value of A is erased.

A variable may be used throughout a program. The computer will use the assigned value wherever the variable occurs. For instance, if A has the value 7, then the expression

```
A+5
```

is evaluated as 7+5, or 12. The expression

```
3*A-10
```

is evaluated as 3*7-10, or 21-10, or 11. The expression

```
2*A^2
```

is evaluated as 2*7^2, or 2*49, or 98.

TEST YOUR UNDERSTANDING 1 (answer on page 43)

Suppose that A has the value 4 and B has the value 3. What is the value of the expression A*2+2*B^2?

NOTE: If you use a variable whose value you have not set, ATARI BASIC will automatically assign it the value 0.

Here are three useful programming hints:

1. The word LET is optional. For example, the statement

   ```
   LET A=5
   ```

   may be abbreviated

   ```
   A=5
   ```

2. As we mentioned earlier, several statements may be included on one line. To do so, just separate the various statements by colons. In particular, a single line may be used to assign values to several variables. For instance, the line

```
100 LET C=18: LET D=23: LET E=2.718
```

assigns C the value 18, D the value 23, and E the value 2.718. Using shortcut #1., we may write this instruction in the simpler form

```
100 C=18:D=23:E=2.718
```

3. As we also mentioned earlier, you may use statements that extend beyond a single display line. This is especially useful when assigning values to many variables as in shortcut #2 above. When you reach the end of a display line, just keep on typing. The computer will automatically jump to the next display line. Hit RETURN when you have finished the program line. Remember that a program line may be up to three display lines long (normally 114 characters). If you type more than three lines, the computer will ignore everything beyond the first three display lines. After 107 characters, the 400 and 800 make a beeping sound. (The XL computers cause the TV to beep.) This is a warning that the end of the program line is approaching. The same thing happens in the direct mode.

Variables may also be used in PRINT statements. For example, the statement

```
10 PRINT A
```

will cause the computer to print the current value of A (in the first print zone). The statement

```
20 PRINT A,B,C
```

will result in printing the current values of A, B, and C in print zones 1, 2, and 3, respectively.

TEST YOUR UNDERSTANDING 2 (answer on page 43)

Suppose A has the value 5. What will the result of the following line be?

```
10 PRINT A,A^2,2*A^2
```

Example 1. Consider the three numbers 5.71, 3.23, and 4.05. Calculate their sum, their product, and the sum of their squares (that is, the sum of their second powers; such a sum is often used in statistics).

Solution. Introduce variables A, B, and C and set them equal to the three numbers, respectively. Then compute the desired quantities.

```
10 LET A=5.71:LET B=3.23:LET C=4.05
20 PRINT "THE SUM IS "; A+B+C
30 PRINT "THE PRODUCT IS "; A*B*C
40 PRINT "THE SUM OF THE SQUARES IS ";
   A^2+B^2+C^2
50 END
```

TEST YOUR UNDERSTANDING 3 (answer on page 43)

Consider the numbers 101, 102, 103, 104, 105, and 106. Write a program that calculates the product of the first two, the first three, the first four, the first five, and all six numbers.

The following imagery may help you understand how ATARI BASIC handles variables. When ATARI BASIC first encounters a variable, let's say A, it sets up a box (actually one or more memory locations) which it labels "A" (see Figure 2-1). In this box it stores the current value of A. When you request a change in the value of A, the computer throws out the current contents of the box and inserts the new value.

Notice that the value of a variable need not remain the same throughout a program. At any point in the program, you may change the value of a variable (with a LET statement, for example). If a program is called upon to evaluate an expression involving a variable, it will always use the current value of the variable, ignoring any previous values the variable may have had at earlier points in the program.

TEST YOUR UNDERSTANDING 4 (answer on page 43)

Suppose that a loan for $5,000 has an interest rate of 1.5 percent on the unpaid balance at the end of each month. Suppose that at the end of the first month, you make a payment of $150 (after the interest is added). Write a program to calculate the balance at the end of the first month. Design your program to calculate the balance after the payment. (Begin by letting B = the loan balance, I = the interest, and P = the payment. After the payment, the new balance is B+I-P.)

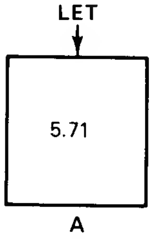

Figure 2-1.The variable A.

Example 2. What will the output of the following program be?

```
10 LET A=10:LET B=20
20 LET A=5
30 PRINT A+B+C,A*B*C
40 END
```

Solution. Note that no value for C is specified, so C=0. Also note that the value of A is initially set to 10. However, in line 20, this value is changed to 5. So in line 30, A, B, and C have the respective values 5, 20, and 0. Therefore, the output will be

```
25          0
```

To the computer, the statement

```
LET A=
```

means that the current value of A is to be replaced by the current value of whatever appears to the right of the equal sign. Therefore, if we write

```
LET A=A+1
```

we are asking the computer to replace the current value of A with the current value of A + 1. In other words, we are asking the computer to increase the value of A by 1. So if the current value of A is 4, the value of A after performing the instruction is 4 + 1, or 5. If the current value of A is 2.351, the value of A after performing the instruction is 2.351 + 1, or 3.351. As you will see later, this statement is used frequently.

TEST YOUR UNDERSTANDING 5 (answer on page 43)

What is the output of the following program?

```
10 LET A=5
20 LET A=A+1
30 LET A=2*A
```

```
40 LET A=A+B
50 PRINT A
60 END
```

Legal Variable Names

As we mentioned previously, you may use any letter of the alphabet as a variable name. ATARI computers are quite flexible concerning variable names. A sequence of characters is a legal variable name if it satisfies the following:

1. It must begin with a capital letter.
2. Each of its characters must be a capital letter or one of the digits 1, 2, 3, 4, 5, 6, 7, 8, 9, or 0.
3. It cannot be any sequences of characters which are reserved by ATARI BASIC for special meanings. Examples of such words are

   `IF, ON, OR, THEN, GOTO, RUN, and LIST`
4. A variable can be up to 112 characters long (that's larger than anyone will probably ever need!!!).

Here are some examples of legal variables you can use:

`A, A1, A1B2, SCORE, DELAY1, TAX, BALANCE`

The sequences 1A and a1 are not legal variable names since neither begins with a capital letter. The sequence DELAY 1 is not legal since it contains a space (sometimes called the **empty character**).

String Variables

So far, all the variables we have discussed have represented numerical values. ATARI BASIC also allows variables for strings. Such variables are called **string variables** and are denoted by a variable name followed by a dollar sign. Thus, A$, B1$, and ZZ$ are all valid names of string variables.

To assign a value to a string variable, we use the LET statement with the string enclosed in quotation marks. The word LET is optional, but we will use it for maximum clarity. For example, to set A$ equal to the string "BALANCE SHEET", we can use either the statement

```
LET A$="BALANCE SHEET"
```

or

```
A$="BALANCE SHEET"
```

Before assigning any value to a string variable, you must tell the computer how many characters the string will hold. By the following count

```
BALANCE SHEET
1234567891111
         0123
```

we see that the string "BALANCE SHEET" has a length of 13. If you want to use A$ for this string, you must tell the computer the length, or dimension, of A$ by the following DIM statement:

```
DIM A$(13)
```

This instructs the computer to reserve enough space in its memory to store the value you assign to A$. If A$ will be given several values during a program, use the length of the largest value it will have. If you don't know what the largest value will be, use a number large enough to handle anything that is likely to come along. For example:

```
DIM A$(100)
```

will probably handle most strings you will ever need. After you have dimensioned a string variable and assigned it a value, you can have the computer print its value, just as you would with a numeric variable . Consider this example:

```
10 DIM A$(13)
20 A$="BALANCE SHEET"
30 PRINT A$
40 END
```

Running this program will result in the following screen output:

```
BALANCE SHEET
```

Example 3. What will the output of the following program be?

```
10 DIM A$(16),B$(16)
20 LET A$="MONTHLY RECEIPTS":B$="MONTHLY
   EXPENSES"
30 LET A=20373.19:B=17584.31
40 PRINT A$,,B$
50 PRINT A,,B
60 END
```

Solution.

```
MONTHLY RECEIPTS      MONTHLY EXPENSES
20373.19              17584.31
```

Notice we used the variables A and A$ in the same program. The variables A and A$ are considered different by the computer. Notice also that we used two commas between A and B in line 50. The reason is that "MONTHLY RECEIPTS" required more than one print zone, whereas "20373.19" did not. The extra comma caused the computer to skip a complete print zone and resulted in "17584.31" being lined up with "MONTHLY EXPENSES".

CAUTION: You can only DIMension a variable ONCE in a program. For this reason, don't use DIM statements inside of loops. A good place to put DIMension statements is at the very beginning of a program.

Remarks in Programs

It is very convenient to have remarks in programs that do nothing but explain what the program does. For one thing, remarks make programs easy to read. Remarks also assist in finding errors and making modifications in a program. To insert a remark in a program, you may use the **REM** statement. For example, consider the line

```
10 LET X=5:REM -Initial value of X
```

Whenever the computer encounters REM, it ignores the rest of the line. Thus in the above statement, the computer sets X equal to 5, and then goes on to the next line. The remark serves only to inform anyone reading the program. It should be clear that a REM must be the last (or only) statement on a program line. REM may be abbreviated to R. or . if you wish. Thus, all the following have the same meaning:

REM Initial value of X
R. Initial value of X
. Initial value of X

The REM statement can be conveniently used to temporarily delete a line or part of a line from a program. You might wish to do this, for example, if you are looking for errors in a program or if you just want to see how the program works with the deletion.

To delete a line, just type REM, R., or . right after the line number. If you later wish to restore the line to the program, you have only to remove the REM (or its abbreviation).

TEST YOUR UNDERSTANDING 6 (answer on page 43)

What is the result of the following program line?

```
10 LET A=19: PRINT "A=";A:REM A=age
```

Exercises (answers on page 247)

In Exercises 1-4, explain what is wrong with the given program.

1.
```
10 LET S$="SCORE":LET S=10
20 PRINT S$,S
30 END
```

2.
```
10 LET A=5.1:LET B=3.2
20 PRINT (A+B)/C
30 END
```

3.
```
10 DIM N$(4)
20 LET N$=NAME
30 PRINT N$,"ATARI"
40 END
```

4.
```
10 DIM T$(10)
20 LET T$="TOTAL PRICE"
30 LET T=101.40+219.95+19.98
40 PRINT T$;"=";T
50 END
```

ANSWERS TO TEST YOUR UNDERSTANDINGS 1, 2, 3, 4, 5, and 6

1:
```
26
```

2:
```
5          25            50
```

3:
```
10 LET A=101:LET B=102:LET C=103:LET
   D=104:LET E=105:LET F=106
20 PRINT "THE PRODUCT OF THE FIRST TWO
   IS ";A*B
```

```
   30 PRINT "THE PRODUCT OF THE FIRST
      THREE IS ";A*B*C
   40 PRINT "THE PRODUCT OF THE FIRST
      FOUR IS ";A*B*C*D
   50 PRINT "THE PRODUCT OF THE FIRST
      FIVE IS ";A*B*C*D*E
   60 PRINT "THE PRODUCT OF ALL THE
      NUMBERS IS ";A*B*C*D*E*F
   70 END

4: 10 LET B=5000
   20 LET I=.015*B
   30 LET P=150
   40 LET B=B+I-P
   50 PRINT "THE BALANCE AFTER THE FIRST
      PAYMENT IS ";B
   60 END

5: 12

6: A=19
```

2.5 Error Messages

At some time or another while using the computer, you will probably receive an ERROR message. This is perfectly normal. It is the computer's way of telling you that you typed something it did not understand. In this section, we will discuss the meanings of these messages.

Let's begin with error messages in the direct mode. Type the following with the BASIC cartridge inserted and hit RETURN.

```
KFKR
```

On the screen, the computer will display

```
ERROR- KFKR■
```

This error message informs you that what you typed can't be interpreted by the computer in any way. A cursor-like symbol is located at the end of the line, which is the place where the computer detected the first error in the line. Up to that point, what you typed could have been the first part of a statement assigning a value to the variable KFKR, so the computer had no reason to give you an error message at a prior point.

Here is an example of a statement that will give an error message with a number:

```
PRINT 10^100
```

When you hit RETURN, the computer will display:

```
ERROR- 11
█
```

The 11 is the error number meaning you used a number too large for BASIC to work with. (The largest allowable number is 10^98.)

ATARI computers have numbers for many types of errors. For a complete list of the error numbers and their descriptions, see Appendix A. Most of the descriptions will probably not make much sense to you now, but as you learn more statements, the error messages will become clearer and more helpful.

Now let's look at some error messages you might encounter while typing in a program. Type the line

```
10 PRINT "HELLO":KFKR:PRINT "GOODBYE"
```

and hit RETURN. When you do so, the screen will display

```
10 ERROR- PRINT "HELLO":KFKR█PRINT "GOODBYE"
```

The number 10 in front of the word ERROR informs you that the error was made in line 10. The position of the cursor-like symbol indicates the first place an error was encountered.

Some error messages are not given until the program is run. For example, consider the program

```
10 PRINT 10^100
```

When you type this in, the computer will accept it without giving an error message of any kind. But when you run the program, the computer displays

```
ERROR- 11 AT LINE 10
█
```

As we mentioned earlier, error number 11 tells you that the computer has encountered a number larger than 10^98. The message also tells you the line number at which the error was detected. This is a valuable piece of information when you are running a large program. It gives you a logical starting point for tracking down mistakes in the program. In this case, we would, of course, look for our error in line 10. In other cases, the computer might give us the number of a line which is totally correct. Here is an example of this:

```
10 LET A=8*10^97
20 PRINT A*10
30 END
```

By themselves, lines 10 and 20 are perfectly fine. But when the program is run, the computer displays

```
ERROR- 11 AT LINE 20
■
```

In the above example, we might have wanted line 10 to read

```
10 LET A=8*10^87
```

but made a typing error. The error message said the error was at line 20, but it actually was at line 10. Since error 11 means a number is too large, we could deduce that A*10 was too large, and thus find the error in line 10. This is the hardest kind of error to correct in large programs, since you may have to search the entire program to locate the error.

3

More on ATARI BASIC

3.1 The GOTO Statement

In this chapter we will continue our introduction of ATARI BASIC. Our discussion will center on the instructions for controlling the order of statement execution.

We said earlier that unless instructed to do otherwise, the computer executes the statements in a program in increasing line number order. There are several ways of instructing the computer to deviate from that order. The most basic is by using the GOTO statement. (This is not a typographical error. There is no space between GO and TO.) This instruction has the form

```
GOTO X
```

where X is the number of the line you want the computer to execute next. For example, the instruction

```
1000 GOTO 300
```

will instruct the computer to execute line 300 immediately after line 1000, even if line 300 has already been executed. After executing line 300, the computer will continue in the normal fashion from that line: It will execute the next line after line 300, unless instructed by line 300 to do otherwise. Similarly, the instruction

```
230 GOTO 300
```

will send the computer forward to line 300. If there are any lines between 230 and 300, they will not be executed between the time 230 is executed and the time 300 is executed. The only way intermediate lines could be executed is for some other statement to instruct the computer to go to one of them.

To see how these ideas work, consider the following example:

```
10  LET A=5
20  GOTO 100
30  PRINT
40  PRINT "OUTPUT COMPLETE"
50  END
100 PRINT A
```

```
110 PRINT A*2
120 PRINT A*3
130 GOTO 30
140 END
```

This is the order in which the computer executes the lines:

10, 20, 100, 110, 120, 130, 30, 40, 50

When the program is run, the computer displays

```
5
10
15

OUTPUT COMPLETE
```

It is a good idea not to have a GOTO statement that sends the computer to a line containing only a REM. The reason is that if you ever give a copy of the program away, the program may be typed in without the REMs to save time. Instead, use a GOTO statement with the number of the first line following the REM.

TEST YOUR UNDERSTANDING 1 (answer on page 51)

In what order will the lines in the following program be executed?

```
10  PRINT "YOUR ORDER:"
20  GOTO 100
30  PRINT
40  PRINT "ORDER COMPLETE"
50  GOTO 200
100 PRINT "1000 pounds fertilizer",
    "$99.50"
110 PRINT "300 pounds limestone",
    "$20.10"
120 PRINT "25 pounds premium grass
    seed","$24.95"
130 PRINT "TAX",,,"$7.23"
140 PRINT "TOTAL",,,"$151.78"
150 GOTO 30
200 END
```

The ability to use the GOTO statement to make the computer jump around in a program is critically important. In the remaining

sections of this chapter we will see how to combine the GOTO statement with other statements. But for now, let's make a simple program with the GOTO statement that illustrates a new idea:

```
10 PRINT "HELLO, EVERYBODY!"
20 GOTO 10
30 END
```

Before you run the program, let's go through it as if we were the computer. We start at line 10, which instructs us to print HELLO, EVERYBODY! on the screen. Then we proceed to line 20, which tells us to GOTO line 10. We faithfully follow the instructions and return to line 10. We are now at line 10 for the second time! We print HELLO, EVERYBODY! on the screen a second time, and proceed to line 20. But line 20 sends us back to line 10 again. We could go on, but by now you probably get the idea: HELLO, EVERYBODY will be printed over and over, because line 20 keeps sending us back to line 10.

When a sequence of lines in a program would be executed over and over without anything in the program to stop it, we call the sequence an INFINITE LOOP. Now run the program and watch. Watch some more. Keep on watching. The computer repeatedly prints HELLO, EVERYBODY! on the screen. You might be wondering how to stop the program. It's very simple. Just hit the BREAK key in the upper right hand corner of the keyboard. This key will interrupt the program currently in progress and display on the screen

```
STOPPED AT LINE X
```

where X is the number of the line the computer was executing at the time you hit BREAK. The computer will finish the line it's on, but will not continue to the next line when you hit BREAK.

When it stops, the computer returns to direct mode. It is then ready to accept a command from the keyboard. The program in memory will not be harmed in any way when you hit BREAK. You may have the program pick up where it left off by typing

```
CONT
```

and hitting return. CONT should remind you of "continue." In some cases, this will not cause the program to resume operation. In those cases, you can always execute the program again by simply typing RUN and hitting RETURN. You may also have your program printed on the screen by typing LIST.

Occasionally you may intentionally run a program with an infinite loop. You might do this, for example, if you wanted to repeat-

edly print something on the screen, or if you wanted some process to continue until you decide to discontinue it by hitting BREAK. There will be many examples of such loops later in the book. For example, we will use a statement like

```
100 GOTO 100
```

at the end of a program when we want to keep the display on the screen for an indefinite period of time. However, if an unwanted infinite loop slips into a program, a lot of time could elapse before it is discovered. This is frustrating, and could be expensive if rented equipment is being used. It is therefore a good idea to be on the watch for unwanted infinite loops.

Exercises (answers on page 247)

1. In what order will the lines in the following program be executed?

```
10  LET A=5
20  GOTO 200
30  END
40  PRINT "WHAT IS THE VALUE OF A?"
50  GOTO 30
200 PRINT "A+9=";A+9
210 GOTO 40
220 END
```

2. In what order will the lines in the following program be executed? What will the output be?

```
10 LET A=5
20 GOTO 10*A
30 PRINT 30
40 END
50 PRINT 50
60 GOTO 40
70 END
```

3. In what order will the lines in the following program be executed?

```
10  LET A=2
20  GOTO 100*A
30  PRINT "PRINT"
200 PRINT A
210 LET A=3
220 GOTO 20
300 PRINT A
```

```
310 LET A=4
320 GOTO 20
400 PRINT A
500 END
```

4. Describe what the following program does.

```
10 PRINT 1;
20 GOTO 10
30 END
```

5. a. Write a program that prints an unending sequence of A's on the screen, all in a vertical line.
   b. Alter the program in part a. so that there are two vertical lines of A's separated by four spaces.

ANSWER TO TEST YOUR UNDERSTANDING 1

1: 10, 20, 100, 110, 120, 130, 140, 150, 30, 40, 50, 200

3.2 Doing Repetitive Operations

Suppose we wish to solve 50 similar multiplication problems. It is certainly possible to type in the 50 problems one at a time and let the computer solve them, but this is a very inefficient way to proceed. Suppose that instead of 50 problems there were 500, or even 5000. Typing the problems one at a time would not be practical.

If we can describe to the computer the entire class of problems we want solved, then we can instruct the computer to solve them using only a few statements. Let us consider a concrete problem. Suppose we wish to calculate and print the quantities

```
7*1, 7*2, 7*3, ... , 7*10
```

That is, we wish to calculate and print the first ten multiples of 7. This calculation can be described to the computer as calculating 7*N, where the variable N is allowed to assume, one at a time, each of the values 1,2,3,...,10. Here is a sequence of statements that instructs the computer to carry out the calculations:

```
10 FOR N=1 TO 10
20    PRINT  7*N  <--{This instruction
30 NEXT N          {repeated 10 times
40 END
```

The sequence of statements 10, 20, 30 is called a **loop.**

When the computer first encounters the FOR statement, it sets N equal to the first value after "FOR N = ", in this case 1, and proceeds to the next line. Line 20 calls for printing 7*N. Since N is equal to 1, we have 7*N=7*1=7. Therefore, the computer will print a 7. Next comes line 30, which contains a NEXT statement. This increases N by 1 (making it 2) and tells the computer to go back to the corresponding FOR statement in line 10. This next time through the loop, 7*N=7*2=14, so line 20 instructs the computer to print 14. At line 30 the computer increases N again (making the new value 3). Then it returns to the FOR statement in 10 and repeats instruction 20. This time, 7*N=7*3=21, so line 20 prints 21. Line 30 instructs the computer to increase N to 4 and go back to line 10, and so forth. Each time the computer returns to line 30, it increases the value of N by 1. Lines 10, 20, and 30 are repeated 10 times! After the computer "cycles" through the loop 10 times, it will get to line 30 and increase N to 11. Since this number is bigger than 10, it will leave the loop and go on to line 40.

A finite loop is like an infinite loop, in that some sequence of program lines is repeated. However, in an infinite loop there is nothing to stop the repetition, whereas in a finite loop the repetition stops after a certain number of steps.

Type in the above program and type the RUN command. The output will look like this:

```
7
14
21
28
35
42
49
56
63
70
```

The variable N is called the **loop variable.** You may use it inside the loop just as you would use any other variable. For example, it may be used in algebraic calculations and PRINT statements. However, you should not change its value inside the loop. This could mess up the loop. Consider the following program:

```
10 FOR A=1 TO 10
20    LET A=11
30    LET B=B+1
40 NEXT A
50 END
```

At line 10, A is set equal to 1. At line 20, A is set equal to 11. When the program gets to line 40, the computer exits from the loop because A is greater than 10. The program then ends, with the value of B equaling 1. If there would be no line 20, B would equal 10 at the end of the program.

Some Cautions

Here are two of the errors you are most likely to make in dealing with loops.

1. Every FOR statement must have a corresponding NEXT statement. Otherwise ATARI BASIC will not return to the FOR statement. It will go through the loop only once, using the first value of the variable.
2. Be sure the loop variable is not already used with some other meaning. For example, suppose the loop variable N is used before the loop begins. Then the loop will destroy the old value of N and there is no way to get it back after the loop is over.

Let us give you two more examples of programs containing loops. In each case, first decide what you think the computer will print on the screen. Then type in and run the program.
First program:

```
10 FOR N=1 TO 20
20   PRINT "HELLO"
30 NEXT N
40 END
```

Second program:

```
10 FOR N=1 TO 1000
20   PRINT N,:REM -Don't forget the comma-
30 NEXT N
40 END
```

TEST YOUR UNDERSTANDING 1 (answers on page 64)

a. Devise a loop allowing N to assume the values 3 to 20.
b. Write a program which calculates and prints 7*N, with N ranging from 3 to 20.

Making Loops More Readable

Notice that in each of the above programs we have indented the textual portion of line 20. This lets us clearly see the beginning and end of the loop. You can use the TAB key to create the same effect when typing in programs, but when the computer lists the program, all indentations will be deleted. Also, when you list your programs, all abbreviations except ? (one of the two abbreviations for PRINT) will be "un-abbreviated." We will be introducing many new statements which can be abbreviated. Appendix E gives a list of descriptions and abbreviations for all the statements we will cover.

Let's modify our program for printing multiples of 7 to include on each line of output not only 7*N, but also the value of N. To make the table easier to read, let's also add two column headings. The new program reads

```
10 PRINT "N","7*N":PRINT
20 FOR N=1 TO 10
30   PRINT N,7*N
40 NEXT N
50 END
```

The output now looks like this:

```
N          7*N


1          7
2          14
3          21
4          28
5          35
6          42
7          49
8          56
9          63
10         70
```

Notice that there is a PRINT statement at the end of line 10 in the program. Since there is nothing given after PRINT for the computer to print, the computer prints nothing. This results in a line being skipped, and has the effect of separating the column headings from the numbers in the columns.

TEST YOUR UNDERSTANDING 2 (answer on page 64)

What would happen if we changed the number of line 10 to 25 in the preceding program?

Let us now illustrate some of the many uses loops have by means of some examples.

Example 1. Write a program to calculate 1+2+3+...+100.

Solution. Let us use a variable S (first letter of "sum") to contain the sum. Let us start S at 0 and use a loop to successively add to S the numbers 1,2,3,...,100. Here is the program:

```
10 LET S = 0
20 FOR N = 1 TO 100
30   LET S = S+N <------ {This instruction
40 NEXT N                {repeated 100 times
50 PRINT "1+2+3+...+100=";S
60 END
```

When we enter the loop the first time, S=0 and N=1. Line 30 then replaces S by S+N, or 0+1. Line 40 increases N to 2 and sends us back to line 20. In line 30, S (which is now 0+1) is replaced by S+N, or 0+1+2. Line 40 now increases N to 3 and sends us back to line 20. Line 30 then sets S equal to 0+1+2+3. Finally, on the 100th time through the loop, S is replaced by 0+1+2+ ... +100, the desired sum. If we run the program, the output will be:

```
1+2+3+...+100=5050
```

TEST YOUR UNDERSTANDING 3 (answers on page 64)

a. Write a program to calculate 101+102+...+110.
b. Write a program to calculate and display the numbers 2, 2*2, 2*3, ..., 2*20

Example 2. Write a program to calculate the sum:

```
1*2+2*3+3*4+...+49*50
```

Solution. We let the sum be contained in the variable S, as we did in Example 1. The quantities to be added are just the numbers N*(N+1) for N=1, 2, 3, ..., 49. Here is our program:

```
10 LET S = 0
20 FOR N = 1 TO 49
30   LET S = S + N*(N+1)
40 NEXT N
50 PRINT "1*2+2*3+3*4+...+49*50=";S
60 END
```

Nested Loops

In many applications, it is necessary to execute a loop within a loop. For example, suppose we wish to print the following series of numbers:

```
101, 102, 103, 104, 105, 106, 107, 108, 109
201, 202, 203, 204, 205, 206, 207, 208, 209
301, 302, 303, 304, 305, 306, 307, 308, 309
401, 402, 403, 404, 405, 406, 407, 408, 409
501, 502, 503, 504, 505, 506, 507, 508, 509
```

There are five groups of 10 numbers each. Each line may be printed using a loop. For example, the first line may be printed with

```
100 FOR I=1 TO 9
110   PRINT 100+I;" ";:REM -This leaves a
      space between numbers-
120 NEXT I
```

The second line may be computed with

```
100 FOR I=1 TO 9
110   PRINT 200+I;" ";
120 NEXT I
```

And the last line may be computed with

```
100 FOR I=1 TO 9
110   PRINT 500+I;" ";
120 NEXT I
```

We could compute the desired numbers by repeating essentially the same instructions five times, but it is much easier to do the repetition using a loop. The numbers to be added to I range from 100 (which is 1*100) for the first line, to 500 (which is 5*100) for the last line. This suggests that we represent these numbers as J*100, where J is a loop variable which runs from 1 to 5. We may then print our desired table of numbers using this program:

```
10  FOR J=1 TO 5
100   FOR I=1 TO 9
110     PRINT J*100+I;" ";
120   NEXT I
200   PRINT:REM -This will cause the next
      group to be printed on the next line-
300 NEXT J
400 END
```

The instructions that are indented one level are repeated 5 times, corresponding to the values J=1 through J=5. On the first repetition (J=1), lines 100-120 instruct the computer to print the first group of numbers on the first display line and line 200 causes the computer to jump to the next display line; on the second repetition (J=2), lines 100-120 instruct the computer to print the numbers in the second group on the second display line, and so forth. Note how the indentations help you to read the program. This is an example of good programming style.

If a loop is contained within a loop, we say the loops are **nested.** ATARI BASIC allows you to have nesting in as many layers as you would probably ever need (a loop within a loop within a loop, ...).

TEST YOUR UNDERSTANDING 4 (answer on page 64)

Write a program to print the following table of numbers. Remember to use nested loops!

```
1     11    21    31
2     12    22    32
3     13    23    33
4     14    24    34
5     15    25    35
6     16    26    36
7     17    27    37
8     18    28    38
9     19    29    39
10    20    30    40
```

CAUTION: Be careful not to "overlap" loops. For example, the following sequence should not be used:

```
10 FOR J=1 TO 100
20 FOR K=1 TO 50
.
.
.
80 NEXT J
90   NEXT K
```

Rather, the NEXT K statement must precede the NEXT J, so the K loop is "completely inside" the J loop.

Applications of Loops

Example 3. You borrow $7000 to buy a car. The interest rate is one percent per month. (This means that at the end of every month you are charged interest equal to 1% of the amount you still owe.) Your monthly payments are $232.50. Write a program which computes the balance (amount owed) at the end of each of the first 36 months.

Solution. Let B denote the balance owed. Initially we have B equal to 7000 dollars. At the end of each month, B will change because a month's interest is added and your payment of 232.50 is subtracted. Let I denote the monthly interest. Then I=.01*B since the interest rate is 1% per month. For example, at the end of the first month, the interest owed is .01*7000.00 = $70.00. Let P denote the monthly payment, so that P=232.50. At the end of each month, B changes to B+I-P. Here is a program which performs these calculations:

```
10  PRINT "MONTH","BALANCE"
20  PRINT: REM -Skip a line-
30  LET B=7000: REM B=initial balance
40  LET P=232.50: REM P=monthly payment
50  FOR M=1 TO 36: REM M=month number
60    LET I=.01*B: REM -Calculate interest
      for month-
70    LET B=B+I-P: REM -Calculate new
      balance-
80    PRINT M,"$";B: REM -Print out data for
      month-
90  NEXT M
100 END
```

You should run this program. Notice that it runs, but it is pretty useless because the screen will not contain all of the output. Most of the output goes flying by before you can read it. One method

for remedying this situation is to press CTRL and 1 simultaneously as the output scrolls by on the screen. This will pause execution of the program and freeze the contents of the screen. To resume execution and unfreeze the screen press CTRL and 1 again. The output will begin to scroll again. To use this technique requires some manual dexterity. Moreover, it is difficult to stop the scrolling at a desired location.

TEST YOUR UNDERSTANDING 5

Run the program of Example 3 and practice freezing the output on the screen. It may take several runs before you are comfortable with the procedure.

Another method of adapting the output to your screen size is by printing only 12 months of data at one time.This amount of data will fit since the screen contains 24 lines. We will use a second loop to keep track of 12 month periods. The variable for the new loop will be Y (for "year"), and Y will go from 0 to 2. The month variable will be M as before, but now M will go only from 1 to 12. The month number will now be 12*Y+M since there are twelve months in each year. Here is the revised program:

```
10  LET B=7000
20  LET P=232.50
30  FOR Y=0 TO 2: REM Y=year number
40    PRINT "MONTH","BALANCE"
50    PRINT:REM -Skip a line-
60    FOR M=1 TO 12:REM -Run through the
      months of year Y-
70      LET I=.01*B:REM -Calculate interest
        for month-
80      LET B=B+I-P:REM -Calculate new
        balance-
90      PRINT 12*Y+M,"$";B:REM -Print data
        for month-
100   NEXT M
110   STOP:REM -Halts execution-
120   GRAPHICS 0:REM -Clears screen-
130 NEXT Y:REM -Go to next 12 months-
140 END
```

This program utilizes several new statements. In line 110, we use the STOP statement. This has the same effect as pressing BREAK.

It causes the computer to stop execution of the program. The computer remembers where it stops, and all values of the variables are preserved. The STOP statement also leaves unchanged the contents of the screen. You can take as long as you wish to examine the data on the screen. When you are ready for the program to continue, type CONT and hit RETURN. The computer will resume where it left off. The first instruction it encounters is in line 120. GRAPHICS 0 clears the screen. So, after being told to continue, the computer clears the screen and goes on to the next value of Y for the next 12 months of data. Here is a copy of the output for the ATARI 800, including the statements you type. The output for the 600XL and 800XL would be slightly different.

```
RUN
MONTH      BALANCE

1          $6837.5
2          $6673.375
3          $6507.60875
4          $6340.184837
5          $6171.086685
6          $6000.297551
7          $5827.800526
8          $5653.578531
9          $5477.614316
10         $5299.890459
11         $5120.389363
12         $4939.093256

STOPPED AT LINE 110
CONT
MONTH      BALANCE

13         $4755.984188
14         $4571.044029
15         $4384.255
16         $4195.597013
17         $4005.052983
18         $3812.603512
19         $3618.229547
20         $3421.911842
21         $3223.63096
22         $3023.367269
23         $2821.10094
24         $2616.81195
```

```
STOPPED AT LINE 110
CONT
MONTH        BALANCE

25           $2410.480069
26           $2202.084869
27           $1991.605717
28           $1779.021774
29           $1564.311991
30           $1347.45511
31           $1128.429661
32           $907.213957
33           $683.786096
34           $458.123956
35           $230.205195
36           $7.246E-03

STOPPED AT LINE 110
```

Note that the data in the output is carried out to as many as six digits after the decimal point, even though the problem deals with dollars and cents. We will look at the problem of rounding numbers in Section 10.3. Also note the balance listed for month 36. It is in scientific notation. The -03 indicates that the decimal point is to be moved three places to the left. The number listed is the same as .007246. This means that the final balance is about seven tenths of one cent!

Using Loops to Create Delays

By using a loop we can create a delay inside the computer. Consider the following sequence of instructions:

```
10 FOR N=1 TO 3000
20 NEXT N
30 END
```

This loop doesn't do anything except waste time! Notice that the computer executes lines 10 and 20 three thousand times! While this may seem like a lot of work, it really isn't for a computer. To obtain a feel for the speed at which the computer works, you should time this sequence of instructions. Such a loop may be used to create a delay. For example, when you wish to keep some data on the screen without stopping the program, just build in a delay.

Here is a part of a program which prints two screens of text. A delay is imposed to give the user time to read the first screen.

```
10 PRINT "THIS IS A GRAPHICS PROGRAM TO
   DISPLAY SALES"
20 PRINT "FOR THE YEAR TO DATE"
30 FOR N=1 TO 5000
40 NEXT N:REM -Delay Loop-
50 GRAPHICS 0:REM -Clears screen-
60 PRINT "YOU MUST SUPPLY THE FOLLOWING
   INFORMATION:"
70 PRINT "PRODUCT, TERRITORY, SALESPERSON"
```

Example 4. Use a loop to produce a blinking display for a security system.

Solution. Suppose that your security system is tied in with your computer and the system detects that an intruder is in your warehouse. Let us print out the message:

```
SECURITY SYSTEM DETECTS INTRUDER IN ZONE 2
```

For attention, let us blink this message on and off by alternately printing the message and clearing the screen.

```
10 FOR N=1 TO 2000
20   PRINT "SECURITY SYSTEM DETECTS INTRUDER
     IN ZONE 2"
30   FOR K=1 TO 50
40   NEXT K
50   GRAPHICS 0:REM -Clears the screen-
60 NEXT N
70 END
```

The loop in lines 30-40 is a delay loop to keep the message on the screen for a moment. Line 50 turns the message off, but the PRINT statement in line 20 turns it back on. The message will blink 2000 times.

TEST YOUR UNDERSTANDING 6 (answer on page 64)

Write a program which blinks your name on the screen 500 times, leaving your name on the screen for a loop of length 50 each time.

More About Loops

In most of our loop examples, the loop variable started at one and increased by one with each repetition of the loop. However, it

is possible to have the loop variable start at any number and increase or decrease by any amount. For example, the instructions

```
10   FOR N=13 TO 5000 STEP 2
.
.
.
1000 NEXT N
```

define a loop in which N starts at 13 and increases by 2 with each repetition, so N will assume the values

```
13, 15, 17, 19,..., 4999
```

Similarly, use of STEP .5 in the above loop will cause N to advance by .5 and assume the values

```
13, 13.5, 14, 14.5, 15.5, 16, 16.5, ... ,
5000
```

It is even possible to have a negative step. In this case, the loop variable will decrease with each loop repetition. For example, the instructions

```
10 FOR N=100 TO 19 STEP -1
.
.
100 NEXT N
```

will "count down" from N=100 to N=19 one unit at a time.

TEST YOUR UNDERSTANDING 7 (answers on page 64)

Write instructions to allow N to assume the following sequences of values:

a. `95, 96.7, 98.4,..., 112`
b. `200, 199.5, 199,..., 100`

Exercises (answers on page 248)

In Exercises 1-4 write programs to compute the given quantities.

1. `12+22+32+...+252`
2. `1+.5+1+1.5+2+...+5`
3. `13+23+33+...+103`
4. `1+1/2+1/3+...+1/100`

5. Write a program to compute N^2, N^3, N^4 for N=1,...,12. The format of your output should be as follows:

```
N          N^2          N^3          N^4

1
2
3
.
.
.
12
```

6. Suppose you have a car loan whose current balance is $4,000.00. The monthly payment is $125.33 and the interest is one percent per month. Make a table of the balances for the next 12 months.
7. Suppose you deposit $1,000 on January 1 of each year into a savings account paying 10 percent interest per year. Suppose the interest is computed on January 1 of each year, based on the balance for the preceding year. Calculate the balance in the account for each of the next 15 years.
8. A stock market analyst predicts that Tyro Computers, Inc. will achieve a 20 percent growth in sales in each of the next three years, and that profits will grow at a 30 percent annual rate. Last year's sales were $35 million and last year's profits were $5.54 million. Project the sales and profits for the next three years, based on the analyst's prediction.

ANSWERS TO TEST YOUR UNDERSTANDINGS 1, 2, 3, 4, 6, and 7

1: a.

```
10 FOR N=3 TO 20
.
.
.
30 NEXT N
40 END
```

b.

```
10 FOR N=3 TO 20
20 PRINT 7*N
30 NEXT N
40 END
```

2: The heading

```
        N           7*N
```

would be printed before each entry of the table.

3: a.

```
10 LET S=0
20 FOR N=101 TO 110
30   LET S=S+N
40 NEXT N
50 PRINT "101+102+103+...+110=";S
60 END
```

b.

```
10 FOR N=1 TO 20
20   PRINT 2*N
30 NEXT N
40 END
```

4:

```
10 FOR J=1 TO 10
20   FOR I=0 TO 3
30     PRINT 10*I+J
40   NEXT I
50 NEXT J
60 END
```

6:

```
10 FOR N=1 TO 500
20   PRINT "<YOUR NAME>"
30   FOR K=1 TO 50
40   NEXT K
50   GRAPHICS 0
60 NEXT N
70 END
```

7: a.

```
10  FOR N=95 TO 112 STEP 1.7
.
.
.
100 NEXT N
110 END
```

b.

```
20  FOR N=200 TO 100 STEP -.5
.
.
.
100 NEXT N
110 END
```

3.3 Letting Your Computer Make Decisions

One of the principal features that makes computers useful as problem-solving tools is their ability to make decisions. ATARI BASIC contains instructions that let you ask a question. The computer will determine the answer and take an action that depends on the answer. Here are some examples of questions that the computer can answer:

```
IS A GREATER THAN ZERO?

IS A^2 AT LEAST 200?

IS AT LEAST ONE OF THE VARIABLES A, B OR C
NEGATIVE?

IS A$ THE STRING "YES"?
```

The **IF...THEN** statement lets you ask such questions. It has the form

```
IF <question> THEN <statement or line number>
```

Here is how this statement works:

1. The "question" part of an IF...THEN statement allows you to ask questions like those above.
2. If the answer to the question is YES, the program executes the portion of the statement following THEN.
   a. If a statement follows THEN, that statement is executed.
   b. If a line number follows THEN, the computer jumps to that line and executes it, just as it would with a GOTO statement.
3. If the answer to the question is NO, the computer ignores the portion of the statement following THEN and proceeds to the next line in the program.

For example, consider this instruction:

```
500 IF N=0 THEN PRINT "CALCULATION DONE"
```

The question portion of this instruction is

```
N=0
```

The portion following THEN is the statement

```
PRINT "CALCULATION DONE"
```

When the computer encounters this statement, it first determines whether N is equal to zero. If so, it prints "CALCULATION DONE" and proceeds to the next program line after line 500. However, if N is not zero, the computer goes immediately to the next program line after 500. It ignores the PRINT statement after THEN.

Here is another example:

```
600 IF 2*A<1 THEN 300
```

When the computer reaches this instruction, it will examine the value of 2*A. If 2*A is less than 1, the computer will go to line 300. Otherwise, it will go on to the line that follows line 600.

Consider the following two lines of a program:

```
60 IF A=500 THEN LET S=1:GOTO 80
70 LET S=0
```

If A=500, then S will be set equal to 1 and the computer will advance to line 80. Otherwise, the computer ignores the LET and GOTO statements in line 60 and proceeds to line 70, where S is set equal to 0. The net effect is that S is set equal to either 1 or 0, depending upon whether A=500 or not. The following lines accomplish this also:

```
60 LET S=0
70 IF A=500 THEN LET S=1
```

After IF, you may insert any expression which the computer may test for truth or falsity. Here are some examples:

```
N=0

N>5 (N is greater than 5)

N<12.9 (N is less than 12.9)

N>=0 (N is greater than or equal to 0)

N<=-1 (N is less than or equal to -1)

N<>0 (N is not equal to 0)

A+B<>C (A+B is not equal to C)

A$="STOP" (A$ is the string "STOP")
```

The expression after IF may contain two or more expressions separated by AND or OR. Here are some examples:

N=0 OR A>B (either N=0 or A>B or both)

N>M AND I=0 (both N>M and I=0)

D=100 OR A$="STOP" (either D=100 or A$="STOP" or both)

N=0 AND D=100 AND A$="STOP" (N=0 and D=100 and A$="STOP")

TEST YOUR UNDERSTANDING 1 (answers on page 77)

Write instructions (one or two lines long, beginning with line 10) which do the following:

a. If A equals 5, then print the value of A plus B.
b. If A is less than 5000, then go to line 300. Otherwise, set A equal to 5000.
c. If N is larger than the sum of I and K, then set N equal to the sum of I and K. Otherwise, go to line 300.
d. If X<0 or X>319, then let S=-S.
e. If A$="DELAY" and D is not equal to 24, then go to line 150

The next few examples illustrate some of the possibilities of combining the IF...THEN and GOTO statements.

Example 1. A lumber supply house has a policy that no charge may exceed $1,000, including a 10 percent processing fee and a 5 percent sales tax. A customer orders 150 2x4 studs at $1.99 each, 30 sheets of plywood at $14.00 each, 300 pounds of nails at $1.14 per pound, and two double hung insulated windows at $187.95 each. Write a program which prepares a bill and decides whether the order is over the credit limit.

Solution. Let's use the variables A1, A2, A3, and A4 to denote, respectively, the number of studs, sheets of plywood, pounds of nails, and windows. Let's use the variables B1, B2, B3, and B4 to denote the unit costs of these four items. The cost of the order is then computed as

```
A1*B1+A2*B2+A3*B3+A4*B4
```

We add 10 percent of this amount to cover processing and form the sum to obtain the total order. Next, we compute 5 percent of the last amount as tax and add it to the total to obtain the total amount due. Finally, we determine whether the total amount due is at most $1,000. If it is, we approve the charge. If not, we print out the message: ORDER EXCEEDS $1,000. CHARGE NOT PERMITTED. Here is our program.

```
10  LET A1=150:LET A2=30:LET A3=300:LET
    A4=2:REM -Assign quantities-
20  LET B1=1.99:LET B2=14:LET B3=1.14:LET
    B4=187.95:REM -Assign prices-
30  LET T= A1*B1+A2*B2+A3*B3+A4*B4:REM
    T=total price
40  PRINT "TOTAL ORDER",T
50  LET P=.1*T: REM P=processing fee
60  PRINT "PROCESSING FEE",P
70  LET TX=.05*(P+T): REM TX=sales tax
80  PRINT "SALES TAX",,TX
90  LET DU=T+P+TX: REM DU=Amount due
100 PRINT "AMOUNT DUE", DU
110 IF DU<=1000 THEN 300:REM -Is DU at most
    1000?-
200 PRINT "ORDER EXCEEDS $1,000"
210 PRINT "CHARGE NOT PERMITTED"
220 GOTO 400
300 PRINT "CHARGE APPROVED"
400 END
```

Note the decision in line 110. If the amount due exceeds $1,000, then the computer ignores the THEN part of statement 110 and goes to the next line, line 200, where it prints out a message disallowing the charge. In line 220, the computer is sent to line 400 which is the END of the program. On the other hand, if the amount due is at most $1,000, the computer executes the THEN part of line 110, which instructs the computer to go to line 300, in which credit is approved.

TEST YOUR UNDERSTANDING 2 (answer on page 77)

Consider the following sequence of instructions:

```
100 IF A>=5 THEN 200
110 IF A>=4 THEN 300
120 IF A>=3 THEN 400
```

```
130 IF A>=2 THEN 150
400 END
```

Suppose the current value of A is 3. List the sequence of line numbers which will be executed if the computer starts at line 100.

Example 2. At $20 per square yard, a family can afford up to 500 square feet of carpet for their dining room. They wish to install the carpet in a circular shape. It has been decided that the radius of the carpet is to be a whole number of feet. What is the radius of the largest carpet they can afford? (The area of a circle of radius "R" is PI times R^2, where PI equals approximately 3.14159.)

Solution. Let us compute the areas of the circles of radius 1, 2, 3, 4,... and determine which of the areas are less than 500.

```
10  LET PI=3.14159
20  LET R=1:REM R=radius
30  LET A=PI*R^2:REM A=area
40  IF A>500 THEN 100
50  LET R=R+1:REM -Go to next radius-
60  GOTO 30:REM -Repeat-
100 PRINT "THE LARGEST RADIUS YOU CAN AFFORD
    IS";R-1
110 END
```

Note that line 40 contains an IF...THEN statement. If A, as computed in line 30, is less than 500, the computer ignores the THEN part of line 40 and proceeds to line 50, where R is increased by 1. Then, in line 60, the computer is sent back to line 30 to compute the area again. This process is repeated until a value of R is found for which the area A is greater than 500 (so that a carpet with that radius is too expensive). Then the computer executes the THEN part of line 40, which instructs the computer to go to line 100. Then the computer prints out the radius that is one less than the current radius. (That must be the largest radius that is affordable.) After that, the program ends.

In effect, the lines 30-40-50-60 form a loop. However, we did not use a FOR ... NEXT instruction because we did not know in advance how many times we wanted to execute the loop. We let the computer decide the stopping point with the IF...THEN instruction.

The INPUT Statement

It is very convenient to have the computer request information from you while the program is actually running. This can be accomplished with the INPUT statement. To see how, consider the statement

```
570 INPUT A
```

When the computer encounters this statement in the course of executing the program, it displays

```
?■
```

and waits for you to respond by typing a value for A (and then hitting RETURN). The computer then sets A equal to the value you specified and continues running the program.

You may use an INPUT statement to specify the values of several different variables at one time. These variables may be numeric or string variables. For example, suppose that the computer encounters the statement

```
50 INPUT A,B,C$
```

It will type

```
?■
```

You then type in the desired values for A,B, and C$, in the same order as in the program, separated by commas. For example, suppose you type

```
10.5, 11.42, BEARINGS
```

and hit RETURN. The computer will then set

```
A=10.5, B=11.42, C$="BEARINGS"
```

If you respond to the above question mark by typing only a single number, 10.5, for example, the computer will respond with another question mark and wait for you to give the remaining values. If you attempt to specify a string where you should have a number, the computer will respond with an error message and return to direct mode. You would have to run the program again if you wanted to continue. If you enter a number when the computer expects a string, it will consider the number as a string.

It is helpful to include a prompting message which describes the input the computer is expecting. To do so, first type PRINT and then the message in quotation marks, as you would with a regular PRINT statement. After the message type a semicolon. On the

next line type INPUT, and then the variable you wish to have input. Here is an example:

```
10 DIM A$(100)
20 PRINT "What is your name";
30 INPUT A$
40 END
```

When this program is run, the computer will print

```
What is your name?■
```

and wait for a response. Notice that you do not need to include a question mark in line 20 in the preceding program because the INPUT statement puts a question mark on the screen for you.

TEST YOUR UNDERSTANDING 3 (answer on page 77)

Write a program that allows you to set variables A and B to any desired values with an INPUT statement and print out their sum. Use the program to set the variable A equal to 12 and the variable B equal to 17.

The next two examples illustrate the use of the INPUT statement and provide further practice in using the IF...THEN statement.

Example 3. You are a teacher compiling semester grades. Suppose there are four grades for each student and each grade is on the traditional 0 to 100 scale. Write a program which accepts the grades as input, computes the semester average, and assigns grades according to the following scale:

90-100 A
80-89.9 B
70-79.9 C
60-69.9 D
< 60 F

Solution. We will use an INPUT statement to enter the grades into the computer. Our program will allow you to compute the grades of students, one after the other, via a loop. You may terminate the loop by entering a negative grade. Here is our program.

```
10   PRINT "ENTER STUDENT'S 4 GRADES."
20   PRINT "SEPARATE GRADES BY COMMAS."
```

```
30   PRINT "FOLLOW LAST GRADE WITH RETURN."
40   PRINT "TO END PROGRAM, ENTER A NEGATIVE
     GRADE."
50   PRINT "WHAT ARE THE FOUR GRADES";
60   INPUT A1,A2,A3,A4:REM -Input grades-
70   IF A1<0 OR A2<0 OR A3<0 OR A4<0 THEN
     200:REM -Negative number?-
100  LET A=(A1+A2+A3+A4)/4:REM -Average of
     the four grades-
110  PRINT "SEMESTER AVERAGE ";A:REM -Print
     average on screen-
120  IF A>=90 THEN PRINT "SEMESTER GRADE =
     A": GOTO 10
130  IF A>=80 THEN PRINT "SEMESTER GRADE =
     B": GOTO 10
140  IF A>=70 THEN PRINT "SEMESTER GRADE =
     C": GOTO 10
150  IF A>=60 THEN PRINT "SEMESTER GRADE =
     D": GOTO 10
160  PRINT "SEMESTER GRADE = F": GOTO 10
200  END
```

Note the logic for printing out the semester grades. First compute the semester average, A. In line 120 we ask whether A is greater than or equal to 90. If so, we assign the grade A and go to line 10 to obtain the next grade. In case A is less than 90, the THEN part of line 120 is not executed and the computer goes on to line 130. In line 130, we ask whether A is greater than or equal to 80. If so, then we assign the grade B and go back to line 10. (The point is that the only way we can get to line 130 is for A to be less than 90. So if A is greater than or equal to 80, we know that it lies in the B range.) If A is not greater than or equal to 80, we go to line 140, and so forth. This logic may seem a trifle confusing at first, but after repeated use, it will seem quite natural.

The ON ... GOTO Statement

A very useful variant of the GOTO statement is the **ON ... GOTO** statement. It instructs the computer to go to one of several possible program lines, depending on the value of a variable (A, for example) or an expression (2*A+1,for example). Here is an example:

```
30 ON A GOTO 100, 150, 200, 250
```

Whenever the computer comes to line 30, it will go to line 100 if the current value of A is 1, to line 150 if the current value is 2, to line 200 if the current value is 3, and to line 250 if the current value is 4. There could be a longer list of line numbers after GOTO. If the current value of A is negative or greater than 255, line 30 will cause the computer to give you an error message. If the current value of A is 0 or greater than the number of line numbers after GOTO (but less than 256), the computer will ignore line 30 and go to the next line after line 30. If A is not an integer, the computer will round off before deciding what line number to go to.

Let's see the ON ... GOTO statement in action. We will write a program for maintaining a checkbook. The user will tell the computer what sort of transaction is to be entered by inputting a number A. Depending on the value of A, an ON...GOTO statement will send the computer to the part of the program that handles the desired transaction.

Example 4. Write a program to maintain your checkbook. The program should allow you to record an initial balance, enter deposits, and enter checks. It should also warn you of overdrafts.

Solution. Let the variable B always contain the current balance in the checkbook. The program will first ask for the starting balance. Then it will ask for the type of transaction you wish to record. A "1" will mean that you wish to record a deposit; a "2" will mean you wish to record a check; a "3" will mean that you are finished entering transactions and wish to terminate the program. After entering each transaction, the computer will figure your new balance, report it to you, will check for an overdraft, and report any overdraft to you. In case of an overdraft, the program will allow you to cancel the preceding check. When each transaction is completed, the computer will give you the option of quitting or going on to another transaction.

```
10  DIM E$(1): PRINT "WHAT IS YOUR STARTING
    BALANCE";
20  INPUT B
30  PRINT "1) ADD DEPOSIT"
40  PRINT "2) SUBTRACT CHECK"
50  PRINT "3) QUIT"
60  PRINT
70  PRINT "NUMBER OF OPTION DESIRED (1/2/3)";
80  INPUT A
90  ON A GOTO 110, 210, 500
100 REM -Add deposit-
```

```
110 PRINT "AMOUNT OF DEPOSIT";
120 INPUT D
130 LET B=B+D :REM -Add desposit to balance-
140 PRINT "YOUR NEW BALANCE IS ";"$";B
150 GOTO 30
200 REM -Subtract check-
210 PRINT "AMOUNT OF CHECK";
220 INPUT C
230 LET B=B-C : REM -Deduct check amount-
240 IF B<0 THEN 310 :REM -Test for overdraft-
250 PRINT "YOUR NEW BALANCE IS ";"$";B
260 GOTO 30
300 REM -Process overdraft-
310 PRINT "LAST CHECK CAUSES OVERDRAFT"
320 PRINT "DO YOU WISH TO CANCEL CHECK(Y/N)";
330 INPUT E$
340 IF E$="Y" THEN 410
350 PRINT "YOUR NEW BALANCE IS ";"$";B
360 GOTO 30
400 REM -Cancel check-
410 LET B=B+C :REM -Cancel last check-
420 GOTO 30
500 END
```

You should scan this program carefully to make sure you understand how each of the INPUT and ON...GOTO statements is used. In addition, you should use this program to obtain a feel for the dialog between you and your computer when INPUT statements are used.

Note how the above program is divided into sections. For visual purposes, each section begins with a line number which is a multiple of 100. Moreover, each section begins with a comment which identifies the function of the section. In order to write a complex program, you should break the program into manageable sections. Don't get caught in the common "maze of complexity." Work out one section at a time and include plenty of REMs to clarify each statement. Then put the various sections together into one program.

Example 5. Write a program which tests mastery in addition of two-digit numbers. Let the user suggest the problems, and let the program keep score of the number correct out of ten.

Solution. Let us request the program user to provide pairs of numbers via an INPUT statement. The sum will also be requested via an INPUT statement. An IF ... THEN statement will be used to

judge the correctness. The variable S will keep track of the number correct. We will use a loop to repeat the process ten times.

```
10  FOR N=1 TO 10:REM -Loop to give 10
    problems-
20    PRINT "TYPE TWO 2-DIGIT NUMBERS";
30    INPUT A,B
40    PRINT "WHAT IS THEIR SUM";
50    INPUT C
60    IF A+B=C THEN 210
100   REM -Respond to incorrect answer-
110   PRINT "SORRY. THE CORRECT ANSWER IS ";
      "A+B
120   GOTO 300:REM -Go to the next problem-
200   REM -Respond to correct answer-
210   PRINT "YOUR ANSWER IS CORRECT!
      CONGRATULATIONS!"
220   LET S=S+1:REM -Increase score by 1-
300 NEXT N
400 REM -Print score for 10 problems-
410 PRINT "YOUR SCORE IS ";S;" CORRECT OUT OF
    10"
510 PRINT "TO TRY AGAIN, TYPE RUN"
600 END
```

Exercises (answers on page 249)

Here are some exercises that you can try to solve. Of course, you don't have to do them all. Just do the ones which seem interesting or challenging to you. It is important that you move at your own pace.

1. Write a program to calculate and print out all perfect squares which are less than 45,000.(Perfect squares are the numbers 1*1, 2*2, 3*3, 4*4,...)
2. Write a program to determine all of the circles of integer radius and area less than or equal to 5,000 square feet. (The area of a circle of radius R is PI*R^2, where PI=3.14159 approximately.)
3. Write a program to find all positive integers X such that X^3<175,000.
4. Modify the arithmetic testing program of Example 5 so that the operation tested for is multiplication instead of addition.
5. Modify the arithmetic testing program of Example 5 so that it allows you to choose, at the beginning of each group

of ten problems, from among these operations: addition, subtraction, or multiplication.

6. Write a program which accepts three numbers via an INPUT statement and determines the largest of the three.
7. Write a program which accepts three numbers via an INPUT statement and determines the smallest of the three.
8. Write a program which accepts a set of numbers via INPUT statements and determines the largest among them.
9. Write a program which accepts a set of numbers via INPUT statements and determines the smallest among them.
10. A credit card company computes its monthly interest charge on unpaid balances as follows: 1.5 percent on amounts up to $500 and 1 percent on any excess over $500. Write a program that accepts balances as input and then computes the interest charge and new balance.
11. Write a program which does the arithmetic of a cash register. That is, let the program accept purchases via INPUT statements, then total the purchases, figure out sales tax (assume 5 percent), and compute the total purchase. Let the program ask for the amount of payment given and then let it compute the change due.
12. Write a program that analyzes cash flow. Let the program ask for cash on hand as well as accounts expected to be received in the next month. Let the program also compute the total anticipated cash for the month. Let the program ask for the bills due in the next month, and let it compute the total accounts payable during the month. By comparing the amounts to be received and to be paid out, let the program compute the net cash flow for the month and report either a surplus or a deficit.

ANSWERS TO TEST YOUR UNDERSTANDINGS 1, 2, and 3

1: a.
```
10 IF A=5 THEN PRINT A+B
```
b.
```
10 IF A<500 THEN 300
20 LET A=5000
```
c.
```
10 IF N>I+K THEN LET N=I+K
20 GOTO 300
```
d.
```
10 IF X<0 OR X>319 THEN LET S=-S
```
e.
```
10 IF A$="DELAY" AND D<>24 THEN 150
```

2: 100-110-120-400

3:
```
10 PRINT "WHAT ARE THE VALUES OF A AND
   B";
20 INPUT A,B
30 PRINT A;"+";B;"=";A+B
40 END
```

3.4 Planning Your Program

As you proceed with this book, you will notice programs getting longer and more complicated. Pretty soon, you too will be writing programs of this sort yourself. If you do not have a structured method for writing programs, they may contain errors and be hard to debug. The old saying, "A picture is worth a thousand words" is true for computer programming. In designing a program, especially a long one, it helps to draw a picture called a flowchart.

A **flowchart** is an organized chart that lets you plan your program before you begin writing it. A flowchart consists mainly of boxes, each containing a single instruction corresponding to one or more lines of a program. The boxes are connected by arrows that show the order of execution of the instructions. Figure 3-1 is an example of a very simple flowchart.

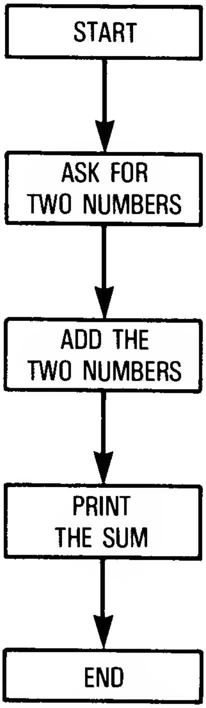

Figure 3-1. A flowchart.

This flowchart is the framework of a simple program:

```
10 PRINT "TYPE TWO NUMBERS TO BE ADDED"
20 INPUT N1, N2
30 LET S=N1+N2
40 PRINT "THE SUM OF ";N1;" AND ";N2;" IS ";S
50 END
```

Of course, a program this simple needs no flowchart, but let's say we wanted the computer to continue asking us values and printing the sum until we hit BREAK. Our new flowchart would look like the one in Figure 3-2.

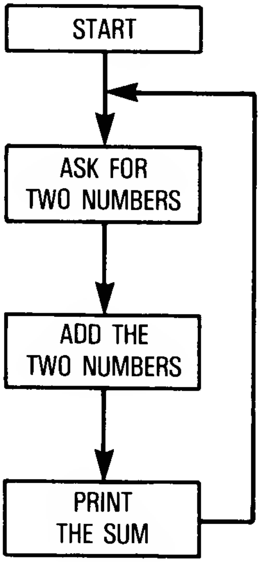

Figure 3-2. Another flowchart.

The arrow in the above flowchart is the equivalent of a GOTO statement in a program. Here is what the program for this flowchart would look like:

```
10 PRINT "TYPE TWO NUMBERS TO BE ADDED"
20 INPUT N1, N2
30 LET S=N1+N2
40 PRINT "THE SUM OF ";N1;" AND ";N2;" IS ";S
50 PRINT
60 GOTO 10
70 END
```

Now let's give an example in which a decision is required. Let's stick with our addition problem, but have the computer ask us for the sum, check it for correctness, and inform us of our perform-

ance. As is commonly done, we will put any step requiring a decision by the computer in a diamond-shaped box. There will be two arrows coming from such a box, depending on the decision made. Our flowchart looks like the one in Figure 3-3. Our program for this flowchart would be:

```
10 PRINT "TYPE TWO NUMBERS TO BE ADDED"
20 INPUT N1, N2
30 PRINT "WHAT DO YOU THINK THE SUM IS";
```

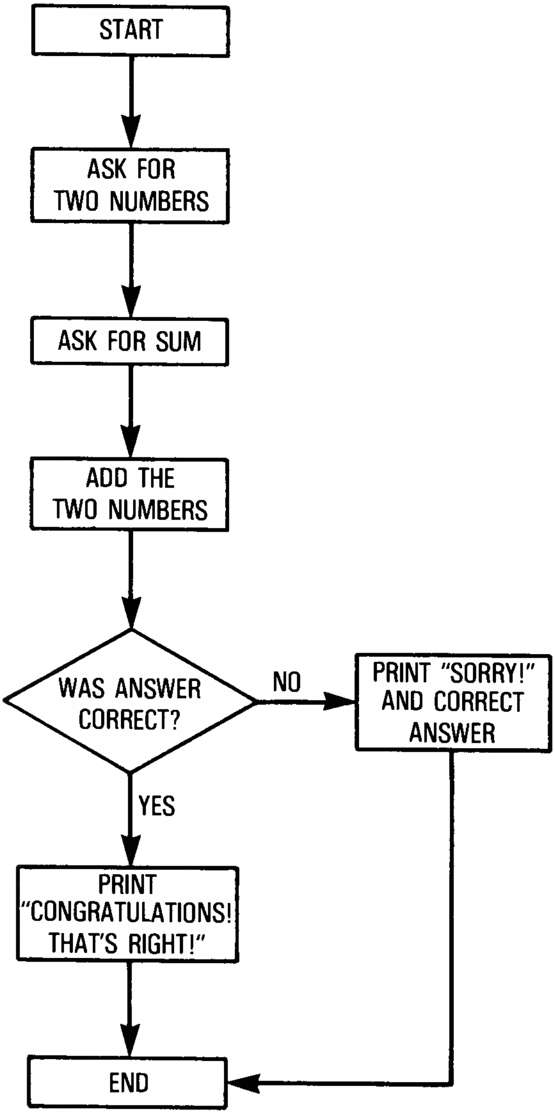

Figure 3-3. A flowchart with a decision box.

```
40 INPUT A
50 LET S=N1+N2
60 IF A<>S THEN PRINT "SORRY! THE SUM OF ";
   N1;" AND ";N2;" IS ";S:GOTO 70
60 PRINT "CONGRATULATIONS! THAT'S RIGHT!"
70 END
```

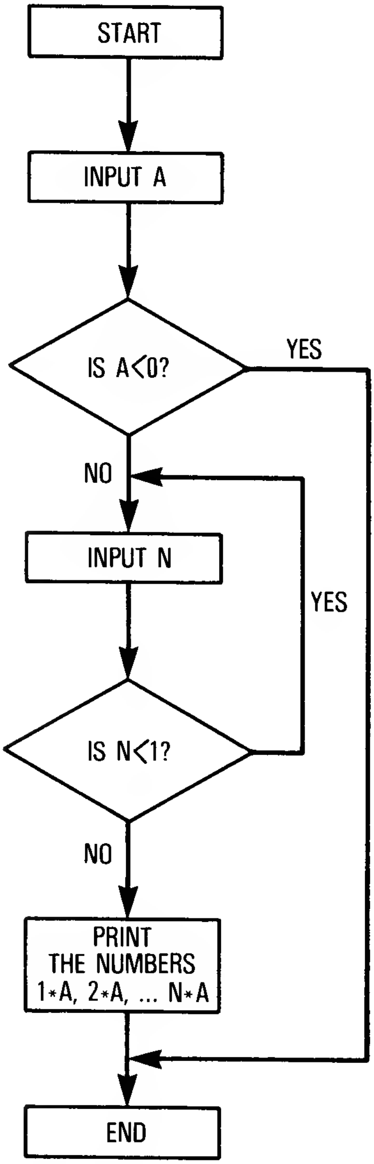

Figure 3-4. Exercise 1 flowchart.

Flowcharting can help you structure and write programs. The flowchart might even help you debug (correct) programs. Remember, one of the most important factors in writing programs is organization.

Exercises (answers on page 253)

1. Write a program that corresponds to the flowchart in Figure 3-4.
2. Draw a flowchart that corresponds to the following program:

```
10 LET N=0
20 LET N=N+1
30 LET S=S+1/N
40 IF S<10 THEN 20
50 PRINT N
60 END
```

3. Draw a flowchart to plan a program that computes and prints the average of any collection of numbers input by the user. Write the program using the flowchart.

3.5 Subroutines

In writing programs it is often necessary to use the same sequence of instructions more than once. It may not be convenient (or even feasible) to retype the set of instructions each time it is needed. Fortunately, ATARI BASIC offers a convenient alternative: the subroutine.

A **subroutine** is a program that is incorporated within another, larger program. The subroutine may be used any number of times by the larger program. Often, the lines corresponding to a subroutine are isolated toward the end of the larger program. This arrangement is illustrated in Figure 3-5. The arrow to the subroutine indicates the point in the larger program at which the subroutine is used. The arrow pointing away from the subroutine indicates that, after completion of the subroutine, execution of the main program resumes at the point at which it was interrupted for the subroutine.

Subroutines are handled with the pair of instructions GOSUB and RETURN. The statement

```
100 GOSUB 1000
```

sends the computer to the subroutine which begins at line 1000. The computer starts at line 1000 and carries out statements in

order. When a RETURN statement is reached, the computer goes back to the main program, starting at the first statement after GOSUB.

A simple example of a subroutine is provided by the following lines, which creates a delay of about two seconds:

```
1000 FOR N=1 TO 1000
1010 NEXT N
1020 RETURN
```

You may call this subroutine at any place in the program as many times as you want by the instruction GOSUB 1000. For example, you might have

```
.
.
.
100 PRINT "THIS PROGRAM WILL HELP YOU LEARN"
110 GOSUB 1000
120 PRINT "HOW TO TYPE"
130 GOSUB 1000
.
```

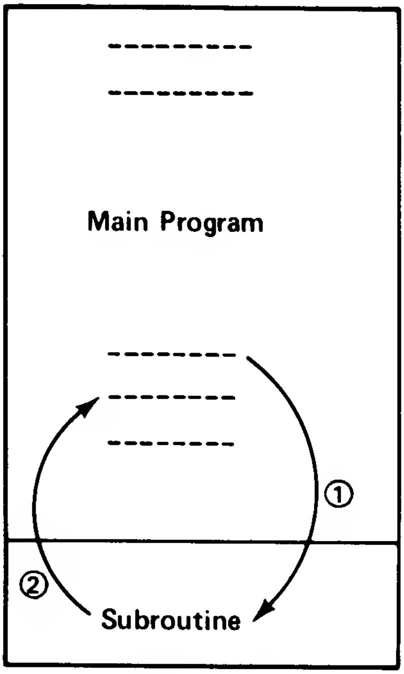

Figure 3-5. A subroutine.

```
.
.
1000 FOR D=1 TO 1000
1010 NEXT D
1020 RETURN
.
.
.
```

At line 100 the computer will print

```
THIS PROGRAM WILL HELP YOU LEARN
```

Then it goes to line 110, which calls the subroutine at line 1000. As a result, the computer will delay about two seconds. Then it returns to line 120 and prints

```
HOW TO TYPE
```

It proceeds immediately to line 130, which calls the two-second delay again. Afterward, the computer continues with the rest of the program.

If you have several subroutines in a single program, it may be difficult to remember their starting line numbers. It sometimes helps to assign suggestive variables to represent the starting line numbers. For the delay subroutine we had at line 1000, you might choose to use DELAY for 1000. Then our last program could take the following form:

```
10  LET DELAY=1000
.
.
.
100 PRINT "THIS PROGRAM WILL HELP YOU LEARN"
110 GOSUB DELAY
120 PRINT "HOW TO TYPE"
130 GOSUB DELAY
.
.
.
1000 FOR D=1 TO 1000
1010 NEXT D
1020 RETURN
.
.
.
```

TEST YOUR UNDERSTANDING 1 (answer on page 88)

Consider the following program:

```
10 GOSUB 40
20 PRINT "LINE 20"
30 END
40 PRINT "LINE 40"
50 RETURN
```

List the line numbers in order of execution.

Nested Subroutines

Just as it is possible to have nested loops, it is also possible to have nested subroutines, that is, a subroutine within a subroutine. Consider the following example:

```
.
.
.
240  GOSUB 1000
.
.
.
1000 PRINT "THE BALANCE IN YOUR ACCOUNT IS
     $";B
1010 GOSUB 2000
1020 RETURN
.
.
.
2000 FOR D=1 TO 3000
2010 NEXT D
2020 RETURN
```

When the computer reaches line 240, it advances to the subroutine starting at line 1000, which contains a PRINT statement. At line 1010 of that subroutine, the subroutine starting at line 2000 is called, so the computer advances to line 2000. It executes the delay subroutine on lines 2000-2020 and then returns to line 1020 to complete the subroutine on lines 1000-1030. Then, the computer is sent by line 1020 back to the next line after line 240. As you can see, the subroutine starting at line 2000 is executed during the exe-

cution of the subroutine starting at line 1000. Thus, the subroutines are nested.

You may use nesting to any level that you are likely to need. (A subroutine within a subroutine within a subroutine, and so forth.) However, you should be aware that a RETURN instruction always refers to the innermost subroutine. To put it another way, a RETURN always refers to the subroutine which was called most recently.

> **CAUTION:** It is possible to accidentally create an infinite nesting of subroutines by repeatedly issuing GOSUB instructions, as in this program:
>
> ```
> 10 PRINT "HELLO"
> 20 GOSUB 10
> ```
>
> The computer will eventually run out of memory trying to keep track of this nesting and an error will result.

The ON ... GOSUB Instruction

The ON...GOSUB statement, like the ON...GOTO statement, allows you to instruct the computer to jump to any of several subroutines, depending on the current value of a variable (A, for example) or an expression (2*A + 1, for example). The form of this statement is

```
ON <expression> GOSUB <line #1>, <line #2>, ...
```

When the computer encounters this instruction, it evaluates <expression> and rounds it off. If the resulting value is 1, the program executes a GOSUB to <line #1>. If the value is 2, the program executes a GOSUB to <line #2>, and so forth. If the value is 0 or more than the number of line numbers provided, the instruction will be ignored. If <expression> is rounded to a negative value or a value larger than 255, an error results.

To illustrate the ON...GOSUB statement, we will rewrite the checkbook program in Section 3.3:

```
10   DIM E$(1)
20   PRINT "WHAT IS YOUR STARTING BALANCE";
30   INPUT B
40   PRINT "1) ADD DEPOSIT"
50   PRINT "2) SUBTRACT CHECK"
60   PRINT "3) QUIT"
70   PRINT
```

```
80   PRINT "NUMBER OF OPTION DESIRED (1/2/
     3)";
90   INPUT A
100  ON A GOSUB 1010, 2010, 3010
110  PRINT
120  PRINT "YOUR NEW BALANCE IS $";B
130  GOTO 40
1000 REM -Add deposit-
1010 PRINT "AMOUNT OF DEPOSIT";
1020 INPUT D
1030 LET B=B+D
1040 RETURN
2000 REM -Subtract check amount-
2010 PRINT "AMOUNT OF CHECK";
2020 INPUT C
2030 B=B-C
2040 IF B<0 THEN GOSUB 4010
2050 RETURN
3000 REM -Quit-
3010 END
4000 REM -Process overdraft-
4010 PRINT
4020 PRINT "LAST CHECK CAUSES OVERDRAFT"
4030 PRINT "DO YOU WISH TO CANCEL CHECK (Y/
     N)";
4040 INPUT E$
4050 IF E$="Y" THEN B=B+C:PRINT "LAST CHECK
     CANCELLED"
4060 RETURN
```

At line 100, the computer will jump to one of three subroutines, depending on the option selected at line 90. In the subroutines starting at line 1010 and 2010, the necessary calculations are made and the computer is returned to line 120 to print the new balance. At line 130 the computer is sent to line 40 to ask for another choice. When option 3 is chosen, the computer jumps to line 3010, which terminates the program. In line 3010, the END statement "cancels" the GOSUB statement, and ends the program. If a check causes an overdraft, it is detected at line 2040 and the computer jumps to the subroutine starting at line 4010. Thus the program contains nested loops.

Exercises (answers on page 256)

1. Consider the instruction

```
10 ON A+2 GOSUB 100, 200, 300, 400, 500
```

To which line (if any) will the computer advance if

a. A=2
b. A=3
c. A=-2
d. A=6
e. A=-4

2. Consider the program

```
10  LET Y=5
20  LET J=3
30  LET S=Y-J
40  ON S GOSUB 100, 200, 300, 400
50  PRINT "HELLO"
60  END
100 RETURN
200 RETURN
300 RETURN
400 RETURN
```

What are the two lines executed immediately after line 40?

3. Suppose you wish to use the checkbook program to add one deposit, subtract two checks (neither of which causes an overdraft), and then quit. List, in the order of execution, the line numbers that would be executed.

ANSWER TO TEST YOUR UNDERSTANDING 1

1: 10, 40, 50, 20, 30

3.6 POKE, PEEK, and TRAP Statements

As we mentioned earlier, the computer stores information in its memory. All information is stored in the form of numbers, and the memory locations are themselves numbered. You can find the number stored in any memory location by using PEEK. The expression

```
PEEK(M)
```

where M is the number of a memory location, gives the number stored in memory location M. For example, if 1 is stored in memory location 752, then PEEK(752)=1. This might be a valuable

piece of information, because whenever 1 is stored in location 752, the cursor is turned off.

You can have the computer store numbers in certain memory locations by means of POKE. The command

```
POKE M,X
```

instructs the computer to store the number X in memory location M. The number X cannot be greater than 255 or less than 0, or an error will result. The statement

```
POKE 752,1
```

instructs the computer to store the number 1 in memory location 752 and, as we said earlier, this causes the cursor to disappear. You may use this command in the direct mode or in a program to turn off the cursor. In games, for example, you may wish to remove the cursor so it will not interfere with whatever you are displaying on the screen. You can turn the cursor back on with the command

```
POKE 752,0
```

IMPORTANT:
1. Never poke aimlessly into memory. You may delete your program.
2. When you turn off your computer, all memory locations are set to their default values.

TEST YOUR UNDERSTANDING 1 (answers on page 90)

a. Instruct the computer to turn off the cursor.
b. Instruct the computer to turn the cursor back on.

Here are some useful memory locations and their default values:

LOCATION	USE
82	Poke a number from 0-39 here to change the left margin of the screen. Default value = 2.
83	Poke a number from 0-39 here to change the right margin of the screen. Default value = 39.

580	If you poke a 1 here, when someone hits SYSTEM RESET the computer will act as if it had been turned off and on, and all data in RAM will be erased. Default value = 0.
764	Tells number of last key pressed. 255 = no key pressed. Default = 255.
838, 839	Poke 166 in 838 and 238 in 839, and all text usually printed on the screen will be sent to the printer instead. Poke 838,244 and 839,241 for normal operation. Default values are 244 and 241.
53770	If you PEEK here, you will get a random integer from 0-255.

The TRAP statement is used only with a program. It has the format

```
TRAP X
```

where X is a line number in the program which the computer will GOTO if you have an error. In the following program, the computer will GOTO line 100 after the error in line 20 occurs:

```
10  TRAP 100
20  GOTO 546
30  END
100 PRINT "AN ERROR HAS BEEN ENCOUNTERED!"
110 END
```

The following program will not give the previous result because the error occurs in line 10, before the TRAP statement.

```
10  GOTO 546
20  TRAP 100
30  END
100 PRINT "AN ERROR HAS BEEN ENCOUNTERED!"
110 END
```

To disable trap (to turn it off), use the statement:

```
TRAP 40000
```

ANSWERS TO TEST YOUR UNDERSTANDING 1

a. `POKE 752,1`
b. `POKE 752,0`

3.7 Debugging Your Programs

Eliminating errors in programs is called **debugging.** Tracking down program bugs (errors) can be a very tricky business and to be good at it, you must be a good detective.

In Section 2.5, we listed some of the clues which ATARI BASIC automatically supplies, namely the error messages. Sometimes, these clues are not enough to locate an error. For example, your program may run without error. It may just not do what it is supposed to. In this case, no error messages will be triggered. In such circumstances you must be prepared to supply your own clues. Here are some techniques.

Inserting Extra PRINT Statements

You may temporarily insert extra PRINT statements into your program. This technique lets you keep track of a variable as your program is executed. If the printed values of the variable are not what you would expect, you could examine all the statements that affect that variable in hopes of locating the bug in the program. Let's consider the following example:

```
10 LET A=91
20 FOR B=1 TO 13
30   LET A=A-B
40 NEXT B
50 LET C=1/A
60 PRINT C
70 END
```

When you run this program, the computer gives you the following error message:

```
ERROR-   11 AT LINE 50
```

Since line 50 involves A, let's put in a PRINT statement to follow the value of A:

```
35 PRINT A
```

If you run the new program, the computer will print

```
90
88
85
81
76
70
```

```
63
55
46
36
25
13
0

ERROR-   11 AT LINE 50
```

Thus the error occurs when A=0. If you now look at line 50, you see what is wrong. You are trying to divide by 0, which is an invalid arithmetic operation. You could now correct the program in any of several ways, depending on what you really want the program to do. After correcting it you could delete the PRINT statement you inserted at line 35.

Inserting STOP Commands

Sometimes your program runs without error messages, but does not perform as you expect it to. In such a case there may be a flaw in your program planning. To debug such a program, you may temporarily insert STOP commands to force halts wherever you wish. This debugging technique may be used in several ways.

1. When the program encounters a STOP instruction, it halts execution and prints out the line number at which the program was stopped. If the program does stop, you will know the instruction just before the STOP was executed. On the other hand, suppose the program continues on its merry way. This tells you the program is avoiding the instructions immediately preceding the STOP. If you determine the reason for this behavior, then you will likely correct a bug.
2. When the program is halted, the values of the variables are preserved. You may examine them to determine the behavior of your program. (See below for more information.)
3. You may insert several STOP instructions. After each halt, you may note the behavior of the program (line number, values of key variables, and so forth). You may continue execution by typing CONT and hitting RETURN. While the program is stopped, you may change lines in the program or set variables to any desired values before giving the CONT command. The program will still pick up where it left off and will use the new values of any reset variables.

Examining Variables in the Direct Mode

When ATARI BASIC stops executing your program, the current values of the program variables are not destroyed. Rather, they are still in memory and may be examined by printing their values on the screen as an indication of program behavior. This is true even if the program is halted by means of a STOP instruction or by hitting BREAK.

Executing Only a Part of a Program

Sometimes it helps to run only a portion of your program. You may start execution at any line using a variation of the GOTO command. For example, to begin execution at line 500, type

```
GOTO 500
```

and hit RETURN. You can even set any variables in the program to any desired values before giving the GOTO command. For example, the command

```
LET A=6:GOTO 500
```

sets the value of A equal to 6 and causes the program to begin running at line 500. When the program resumes at line 500, the initial value of A will be 6.

We have mentioned only a few debugging techniques to get you started. As you gain more experience, you will probably develop some of your own techniques.

4

Working With Data

4.1 Working With Tabular Data

In the preceding chapter, we introduced the notion of a variable and used variable names like

`A, B1,` and `DELAY`

As we will see in this chapter, there are relatively innocent programs which require hundreds or even thousands of variables. To meet the needs of such programs, ATARI BASIC allows the use of so-called **subscripted variables.** Such variables are used constantly by mathematicians and are identified by numbers attached to a letter. For instance, here is a list of 1000 variables as they might appear in a mathematical work:

$A_1, A_2, A_3, \ldots, A_{1000}$

The numbers used to distinguish the variables are called **subscripts.** Likewise, ATARI BASIC allows definition of variables to be distinguished by subscripts. Since the computer has difficulty placing the numbers in the traditional position, they are placed in parentheses on the same line as the letter. For example, the above list of 1000 different variables would be written in ATARI BASIC as

A(1),A(2),A(3),...,A(1000).

The variable A(1) is not the same as the variable A or even A1. You may use all of them in the same program and ATARI BASIC will interpret them differently.

A subscripted variable is really a group of variables with a common letter identification, distinguished by different integer "subscripts." For instance, the above group of variables would constitute the subscripted variable A( ).

It is often useful to view a subscripted variable as a table or array. For example, the subscripted variable A( ) considered above can be viewed as providing a table of values:

A(1)
A(2)
A(3)
.
.
.
A(1000)

As shown here, the subscripted variables define a table consisting of 1000 rows. For each integer J between 1 and 1000, row number J contains a single entry, namely, the value of the variable A(J). The first row contains the value of A(1), the second the value of A(2), and so forth. Since a subscripted variable can be thought of as a table (or array), subscripted variables are often called **arrays.**

The array shown above is a table consisting of 1000 rows and a single column. ATARI computers let you consider more general arrays. For example, consider the following financial table that records the monthly income for January, Febuary, and March from each of four computer stores:

Month	Store #1	Store #2	Store #3	Store #4
January	1258.38	2437.46	4831.90	987.12
February	1107.83	2045.68	3671.86	1129.47
March	1298.00	2136.88	4016.73	1206.34

This table has three rows and four columns. Its entries may be stored in the computer as a set of 12 variables:

A(1,1) A(1,2) A(1,3) A(1,4)
A(2,1) A(2,2) A(2,3) A(2,4)
A(3,1) A(3,2) A(3,3) A(3,4)

This array of variables is very similar to a normal subscripted variable, except there are now two subscripts. The first subscript indicates the row number and the second subscript indicates the column number. For example, the variable A(3,2) is in the third row and second column.

A collection of variables such as that given above is called a **two-dimensional array** or a **doubly-subscripted variable.** Each setting of the variables in such an array defines a tabular array. For example, if we assign the values

A(1,1) = 1258.38, A(1,2) = 2437.46
A(1,3) = 4831.90, and so forth,

we will have the table of earnings for the computer stores.

You must inform the computer of the sizes of the arrays you plan to use into the program. This allows the computer to allocate memory space to house all the values. To specify the size of an array, use a DIM (for dimension) statement. For example, to define the size of the subscripted variable A(J), J = 1, . . ., 1000, we insert the statement

```
10 DIM A(1000)
```

in the program. This statement informs the computer that it should expect variables A(0), A(1), . . ., A(1000) in the program and that it should set aside memory space for 1001 variables. Note that, in the absence of furthur instructions from you, BASIC begins all subscripts at 0. If you wish to use A(0), fine. If not, ignore it.

You need not use the variables defined by a DIM statement. For example, in the case of the DIM statement above, you might actually use only the variables A(1), . . .,A(900). Don't worry about it! Just make sure you have defined enough variables. Otherwise you could get an error message.

In the case of the subscripted variable above, your program might call for the variable A(1001). This will create an error condition. Suppose that this variable is used first in line 570. When you attempt to run the program, the computer will report

```
ERROR-   9 AT LINE 570
```

Moreover, execution of the program will be halted. To fix the error, merely redo the DIM statement to accommodate the undefined subscript.

To define the size of a two-dimensional array, use a DIM statement of the form:

```
10 DIM A(5,4)
```

TEST YOUR UNDERSTANDING 1 (answers on page 100)

Here is an array:

```
12   645.80
148  489.75
589  12.89
487  14.50
```

a. Define an appropriate subscripted variable to store this data.
b. Define an appropriate DIM statement.

It is possible to dimension several arrays with one DIM statement. For example, the dimension statement

```
10 DIM A(1000),B(5),C(5,6)
```

defines the arrays

```
A(0),..., A(1000)

B(0),..., B(5)

C(I,J), I=0,..., 5; J=0,..., 6.
```

We know how to set aside memory space for the variables of an array. We must now take up the problem of assigning values to these variables. We could use individual LET statements, but with 1000 variables in an array, this could lead to an unmanageable number of statements. There are more convenient methods which make use of loops. The next two examples illustrate two of these methods.

Example 1. Define an array A(J), J=1, 2,..., 1000 and assign the following values to the variables of the array:

```
A(1)=2, A(2)=4, A(3)=6, A(4)=8,...
```

Solution. We wish to assign each variable a value equal to twice its subscript. That is, we wish to assign A(J) the value 2*J. To do this we use a loop:

```
10 DIM A(1000)
20 FOR J=1 TO 1000
30   A(J)=2*J
40 NEXT J
50 END
```

Note that the program ignores the variable A(0). Like any variable which has not been assigned a value, it has the value 0.

TEST YOUR UNDERSTANDING 2 (answer on page 100)

Write a program that assigns the variables A(0),..., A(30) the values A(0)=0, A(1)=1, A(2)=4, A(3)=9, ...

When the computer is first turned on, all variables (including those in arrays) are cleared. All numeric variables are set equal to 0, and all string variables are set equal to the null string (the string

with no characters in it). This also occurs if you type NEW or RUN. If you wish to return all variables to this state during the execution of a program, use the command CLR.

For example, when the computer encounters the command

```
570 CLR
```

it will reset ALL the existing variables. Be aware that CLR will also undimension all arrays and string variables. If you wish to reuse an array or string variable after giving the CLR command, you must redimension it. The CLR command can be convenient if, for example, you wish to use the same array to store two different sets of information at two different stages of the program. After the first use of the array, you could prepare for the use of the second by using a CLR command.

Exercises (answers on page 256)

For each of the tables in Exercises 1-4, define an appropriate array and determine the appropriate DIM statement.

1.
```
5
2
1.7
4.9
11
```

2.
```
1.1   2.0   3.5
1.7   2.4   6.2
```

3.
```
1 2 3
```

4.
```
575.00
249.78
174.98
348.70
```

5. Write a program to display the following array on the screen. Use an array to store the numbers in the table. Use a loop with a READ statement mentioned on page 100 to store the numbers in the array.

Dates	Store #1	Store #2	Store #3
1/1-1/10	57,385.48	62,205.34	38,464.34
1/11-1/20	39,486.98	62,238.24	34,256.32
1/21-1/30	45,467.21	62,211.64	37,973.38

6. Write a program that displays the array of Exercise 5 along with totals of the receipts from each store.

ANSWERS TO TEST YOUR UNDERSTANDINGS 1 and 2

1: a. `A(I,J), I = 1,2,3,4, J = 1,2`
   b. `DIM A(4,2)`

2:
```
10 DIM A(30)
20 FOR J=0 TO 30
30   LET A(J)=J^2
40 NEXT J
50 END
```

4.2 Inputting Data

In the preceding section we introduced arrays and discussed several methods for assigning values to the variables in an array. These methods can be tedious for large arrays. Fortunately, ATARI BASIC provides us an alternate method for inputting data.

A given program may need many different pieces of data. You may store the data needed in one or more DATA statements. Here is an example of a data statement:

```
10 DATA 3.457, 2.588, 11234, WINGSPAN
```

Notice that this data statement consists of four data items: three numbers and one string. The items are separated by commas. You may include as many data items in a single DATA statement as the line allows. Moreover, you may include any number of DATA statements in a program and they may be placed anywhere in the program, although a common placement is at the end of the program. Notice we did not enclose the string WINGSPAN in quotation marks. If we had, the quotation marks would become part of the string.

DATA statements may be used to assign values to variables and, in particular, to variables in arrays. To do this, you use one or more READ statements in conjunction with the DATA statements. For example, suppose the above DATA statement appeared in a program. Further, suppose you wish to assign the values

```
A=3.457, B=2.588, C=11234, Z$="WINGSPAN"
```

This can be accomplished with the READ statement

```
100 READ A, B, C, Z$
```

(In the above example, Z$ must be dimensioned.) On encountering a READ statement, the computer will look for a DATA statement. It will then assign values to the variables in the READ statement by taking the values, in order, from the DATA statement. If there is insufficient data in the first DATA statement, the computer will continue to assign values, using the data in the next DATA statement. If necessary, the computer will proceed to the third DATA statement, and so forth. If the computer runs out of data in DATA statements, it will return an error if it encounters another READ statement.

TEST YOUR UNDERSTANDING 1 (answer on page 107)

Assign the following values using READ and DATA statements: A(1)=5.1, A(2)=4.7, A(3)=5.8, A(4)=3.2, A(5)=7.9, A(6)=6.9. (Don't forget the DIM statement.)

The computer maintains an internal pointer which points to the next DATA item to be used. If the computer encounters a second READ statement, it will start reading where it left off. For example, suppose that instead of the above READ statement, we use the two read statements

```
100 READ A, B
200 READ C, Z$
```

Upon encountering the first statement, the computer will look for the location of the pointer. Initially, it will point to the first item in the first DATA statement. The computer will assign the values A=3.457 and B=2.588. The position of the pointer will be advanced to the third item in the DATA statement. Upon encountering the next READ statement, the computer will assign values beginning with the one designated by the pointer, namely C=11234 and Z$="WINGSPAN".

TEST YOUR UNDERSTANDING 2 (answer on page 107)

What values are assigned to A and B$ by the following program?

```
10 DIM C$(6),B$(4)
20 DATA 10, 30, ENGINE, DOOR
30 READ A,B
40 READ C$,B$
50 END
```

The following example illustrates the use of the READ and DATA statements in assigning values to a variable.

Example 1. Suppose the monthly electricity costs of a certain family are as follows:

Jan.	$89.74	Feb.	$95.84	March	$79.42
Apr.	78.93	May	72.11	June	115.94
July	158.92	Aug.	164.38	Sep.	105.98
Oct.	90.44	Nov.	89.15	Dec.	93.97

Write a program that calculates the average monthly cost of electricity.

Solution. Let us unceremoniously dump all of the numbers shown above into DATA statements at the end of the program. Arbitrarily, let's start the DATA statements at line 1000. This allows us plenty of room. To calculate the average monthly cost, we must add up all the monthly costs to get the total cost and divide the total by 12. To do this, let us use T as a variable to represent the total. Initially, T will be 0 and we will add on the monthly costs one at a time. We do this with a loop and a READ statement. Then we divide by 12 and print the answer. Here is the program:

```
10   FOR J=1 TO 12
20     READ C
30     LET T=T+C
40   NEXT J
50   LET A=T/12:REM -Compute average-
60   PRINT "THE AVERAGE MONTHLY COST OF
     ELECTRICITY IS $";A
70   END
```

```
1000 DATA 89.74, 95.84, 79.42, 78.93, 72.11,
     115.94
1010 DATA 158.92, 164.38, 105.98, 90.44,
     89.15, 93.97
```

You may ask why you should write such a complicated program to find the average of twelve numbers. The point is that by changing the numbers in the DATA statements, you can find the average cost for a different year or for a different consumer. The program could be used over and over.

> **NOTE:** You may not use commas in a string with a DATA statement. The computer will interpret the comma as the dividing point of two data entries. For example, this DATA statement will be interpreted as having two strings instead of one:
>
> ```
> 1000 DATA HELLO, ARTHUR
> ```

Restoring Data

In certain applications, you may wish to read the same DATA statements more than once. To do this you must reset the pointer with the RESTORE statement. For example, consider the following program:

```
10 READ A,B
20 RESTORE
30 READ C,D
40 END
50 DATA 2.3, 5.7, 4.5, 7.3
```

Line 10 sets A equal to 2.3 and B equal to 5.7. The RESTORE statement of line 20 moves the pointer back to the first item of data, 2.3. The READ statement in line 30 then sets C equal to 2.3 and D equal to 5.7. Note that without the RESTORE in line 20, the READ statement in line 30 would set C equal to 4.5 and D equal to 7.3.

It is also possible to restore just one line of data in a series of DATA statements. All you have to do is write the number of the line with the DATA statement to restore right after the RESTORE statement. Here is an example:

```
10   READ A, B, C, D
20   RESTORE 1010
30   READ E, F, G
40   RESTORE 1000
```

```
50   READ H, I, J
60   END
1000 DATA 10, 40, 27
1010 DATA 23, 47, 25
```

This program sets A=10, B=40, C= 27, D=23, E=23, F=47, G=25, H=10. I=40, J=27

Example 2. A small business has five employees. Here are their names and hourly wages:

Name	Hourly Wage
1. Julie Axler	7.75
2. Susan Greer	8.50
3. Chico Lee	8.50
4. Aseem Topez	6.00
5. Raul Polanski	6.00

Write a program that accepts as input the hours worked for the current week and calculates the current gross pay along with the amount of Social Security tax to be withheld from each employee's pay. Assume that the Social Security tax amounts to 6.7 percent of gross pay.

Solution. The individual employees will be represented by J, which will assume the values 1, 2, 3, 4, 5. By means of a loop, their names will be read one at a time by a READ statement. When each name is read, the number of hours worked, H(J), will be entered by an INPUT statement. Since ATARI BASIC does not allow for inputting subscripted variables, for each employee, represented by J, we will input H and then let H(J)=H. We will use a loop to make the computations for each employee and print out each employee's name, along with his or her gross wages (G) and Social Security tax (S). The gross wages equal the number of hours worked times the hourly wage rate, and the Social Security tax equals .067 times the gross wages. It will be necessary to read the names twice, so we will use the RESTORE statement. Here is the program:

```
10   DIM N$(15), H(5)
20   FOR J=1 TO 5
30     READ N$, W
40     PRINT "CURRENT HOURS OF "; N$;
50     INPUT H
60     LET H(J)=H
70   NEXT J
```

```
100  PRINT "EMPLOYEE","GR. WAGES",
     "SOC.SEC.TAX"
110  RESTORE
120  FOR J=1 TO 5
130    READ N$, W
140    LET G=H(J)*W
150    LET S=.067*G
160    PRINT N$, G, S
170  NEXT J
180  END
1000 DATA JULIE AXLER, 7.75, SUSAN GREER,
          8.50, CHICO LEE, 8.50
1010 DATA ASEEM TOPEZ, 6.00, RAUL POLANSKI,
          6.00
```

Notice that in line 30 we read both N$ and W, but used only N$ (in line 40). The reason is that in line 130 we needed to read both N$ and W, which made it necessary to arrange the data as we did in the DATA statements at the end of the program, with names alternated with hourly wage rates.

Common Errors

There are two common errors in using READ and DATA statements. First, you may instruct the program to READ more data than is present in the DATA statements. For example, consider the following program:

```
10 DIM A(5)
20 FOR J=1 TO 5
30   READ A:LET A(J)=A
40 NEXT J
50 END
60 DATA 1,2,3,4
```

This program attempts to read five pieces of data, but the DATA statement only has four. In this case, you will receive an error number 6.

A second common error is attempting to assign a string value to an array (like A$(J) for J = 1, 2, 3, 4, 5). Such an array is not permitted by ATARI BASIC.

Exercises (answers on page 257)

Each of the programs in Exercises 1-6 assigns values to the variables of an array. Determine which values are assigned.

1.
```
10  DIM A(10)
20  FOR J=1 TO 10
30    READ A:A(J)=A
40  NEXT J
50  DATA 2, 4, 6, 8, 10, 12, 14, 16, 18, 20
100 END
```

2.
```
10 DIM A(3), B(3)
20 FOR J=0 TO 3
30   READ A,B:A(J)=A:B(J)=B
40 NEXT J
50 END
60 DATA 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7,
   8.8, 9.9
```

3.
```
10 DIM A(3), B(3)
20 FOR J=0 TO 3
30   READ A:A(J)=A
40 NEXT J
50 FOR J=0 TO 3
60   READ B:B(J)=B
70 NEXT J
80 DATA 1, 2, 3, 4, 5, 6, 7, 8
90 END
```

4.
```
10 DIM A(3), B(3)
20 READ A,B:A(0)=A:B(0)=B
30 READ A,B:A(1)=A:B(1)=B
40 RESTORE
50 READ A,B:A(2)=A:B(2)=B
60 READ A,B:A(3)=A:B(3)=B
70 DATA 1, 2, 3, 4, 5, 6, 7, 8
80 END
```

5.
```
10 DIM A(3,4)
20 FOR I=1 TO 3
30   FOR J=1 TO 4
40     READ A:A(I,J)=A
50   NEXT J
60 NEXT I
70 DATA 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
   12
80 END
```

6.
```
10 DIM A(3,4)
20 FOR J=1 TO 4
```

```
30   FOR I=1 TO 3
40     READ A:A(I,J)=A
50  NEXT I
60 NEXT J
70 DATA 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
   12
80 END
```

Each of the programs in Exercises 7-10 contains an error. Find it.

7.
```
10 DIM A(5)
20 FOR J=1 TO 5
30   READ A:A(J)=A
40 NEXT J
50 DATA 1, 2, 3, 4
60 END
```

8.
```
10 DIM A(5)
20 FOR J=1 TO 4
30   READ A:A(J)=A
40 NEXT J
50 DATA 1, A, 2, B
60 END
```

9.
```
10 DIM A$(10)
20 READ A,A$
30 PRINT A$,A
40 DATA AVERAGE,79
50 END
```

10.
```
10 DIM A$(10),B$(10),C$(10),D$(10)
20 READ A$,B$,C$,D$
30 PRINT B$,D$
40 DATA AVERAGE, HIGH SCORE, LOW SCORE
```

ANSWERS TO TEST YOUR UNDERSTANDINGS 1 and 2

1:
```
10 DIM A(6):FOR J=1 TO 6
20   READ A:A(J)=A
30 NEXT J
40 DATA 5.1, 4.7, 5.8, 3.2, 7.9, 6.9
50 END
```

2: A=10, B$="DOOR"

4.3 Generating Data at Random

One of the most interesting and useful features of your computer is its ability to generate events whose outcomes are random. For example, you may instruct the computer to "throw a pair of dice" and produce a random pair of integers between 1 and 6. You may instruct the computer to "pick a card at random from a deck of 52 cards." You may also program the computer to choose a two-digit number "at random."

The source of all such random choices is the **random number generator.** We will begin by explaining what the random number generator is and how to access it. We will then give a number of interesting applications involving computer-assisted instruction and games of chance.

You may generate random numbers using the ATARI BASIC function RND. To explain how this function works, let us consider the following program:

```
10 FOR X=1 TO 500
20   PRINT RND(0)
30 NEXT X
40 END
```

This program consists of a loop which prints 500 numbers, each obtained by RND(0). Each of these numbers is greater than or equal to 0, but less than 1, that is,

```
0 <= RND(0) < 1
```

Each time RND(0) is called (as in line 20 in the preceding program), the computer makes a "random" choice from among the numbers in the indicated range, so it is extremely unlikely that any two of the 500 numbers printed by the above program would be the same. Notice we used 0 in RND(0). We could replace 0 by any other positive number. However, some number (or numeric expression, like A) must appear inside the parentheses.

To obtain a better idea of what we are talking about, you should generate some random numbers using a program like the one above. Unless you have a printer, 500 numbers will be too many for you to look at in one viewing. The following program will generate 30 random numbers, two on each line:

```
10 FOR J=1 TO 15
20   FOR K=1 TO 2
30     PRINT RND(0),
40   NEXT K
```

```
50   PRINT:REM -Skip to the next line-
60 NEXT J
70 END
```

What makes these numbers "random" is that the procedure the computer uses to select them is "unbiased," with all numbers having the same likelihood of selection. Moreover, if you generate a large collection of random numbers, numbers between 0 and .1 will comprise approximately 10 percent of those chosen, those between .5 and 1 will comprise approximately 50 percent of those chosen, and so forth. In this sense, the random number generator provides a uniform sample of the numbers between 0 and 1.

TEST YOUR UNDERSTANDING 1 (answer on page 115)

Assume that RND is used to generate 1000 numbers. Approximately how many of these numbers would you expect to lie between .6 and .9?

The function RND generates random numbers lying between 0 and 1. In many applications, we will require randomly chosen integers lying in a certain range. To obtain them, you may treat the output of the random number generator as you would any other number. In particular, you may perform arithmetic operations on the random numbers generated.

Suppose we wish to generate random integers chosen from among 1, 2, 3, 4, 5, 6. Let us multiply RND(0) by 6, to obtain 6*RND(0). This is a random number between 0 and 6 (not including 6). Next, let us add 1 to this number, getting 6*RND(0)+1, a random number between 1 and 7. To obtain integers from among 1, 2, 3, 4, 5, 6, we must "chop off" the decimal portion of the number 6*RND(0)+1. To do this, we use the INT function. If X is any number, then INT(X) is the largest integer less than or equal to X. For example,

```
INT(5.23)=5, INT(7.99)=7, INT(100.001)=100
```

For a positive number X, INT(X) is just the number you would get by chopping off X at the decimal point, as in the above examples. Be careful in using INT with negative X. The definition we gave is correct, but unless you think things through, it is easy to make an error. For example:

```
INT(-7.4) = -8
```

since the largest integer less than or equal to -7.4 is equal to -8. (Draw -7.4 and -8 on a number line to see the point!)

Let us get back to our random numbers. To chop off the decimal portion of 6*RND(0)+1, we compute INT(6*RND(0)+1). This last expression is a random number from among 1, 2, 3, 4, 5, 6. Similarly, the expression

```
INT(100*RND(0)+1)
```

may be used to generate random numbers from among the integers 1, 2, 3, ..., 100.

TEST YOUR UNDERSTANDING 2 (answer on page 115)

Generate 0 or 1 at random. (This is the computer analogue of flipping a coin: 0 = heads, 1 = tails.) Run this program to generate 50 coin tosses. How many heads and how many tails occur?

Example 1. Write a program which turns the computer into a pair of dice. Your program should report the two numbers rolled on each roll as well as the sum of the two numbers.

Solution. We will hold the value of die #1 in the variable X and the value of die #2 in variable Y. The program will generate values for X and Y at random, and print out the values and the total X+Y.

```
10  DIM A$(1), B$(1)
20  GRAPHICS 0: REM -Clears screen-
30  LET X=INT(6*RND(0)+1)
40  LET Y=INT(6*RND(0)+1)
50  PRINT "LADIES AND GENTLEMEN, BETS
    PLEASE!"
60  PRINT "ARE ALL BETS DOWN(Y/N)";
70  INPUT A$
80  IF A$="N" THEN 50
100 PRINT "THE ROLL IS ";X;", ";Y
110 PRINT "THE WINNING TOTAL IS ";X+Y
120 PRINT "PLAY AGAIN(Y/N)";
130 INPUT B$
140 IF B$="Y" THEN 20
200 PRINT "THE CASINO IS CLOSING. SORRY!"
210 END
```

Note the use of computer-generated conversation on the screen. Note also how the program uses lines 120-140 to allow the player to control how many times the game will be played.

TEST YOUR UNDERSTANDING 3 (answer on page 115)

Write a program which flips a "biased coin." Let it report "heads" one-third of the time and tails two-thirds of the time.

You may enhance the realism of a gambling program by letting the computer keep track of bets as in the following example.

Example 2. Write a program which turns the computer into a roulette wheel. Let the computer keep track of bets and winnings. For simplicity, assume that the only bets are on single numbers.

Solution. A roulette wheel has 38 positions: 1-36, 0, and 00. In our program, we will represent these as the numbers 1-38, with 37 corresponding to 0 and 38 corresponding to 00. A spin of the wheel will consist of choosing a random integer between 1 and 38. The program will start by asking the number of players. For a typical spin of the wheel, the program will ask for bets by each player. A bet will consist of a number (1-38) and an amount bet. The wheel will then spin. The program will determine the winners and losers. A payoff for a win is 32 times the amount bet. The amount of a loss is the amount bet. Each player has an account, stored in an array A(J), J = 1, 2, 3, ... # of players. At the end of each spin, the accounts are adjusted and displayed. Just as in Example 1, the program asks if another play is desired. Here is the program:

```
10  PRINT "NUMBER OF PLAYERS";
20  INPUT N
30  DIM A(N), B(N), C(N), R$(1)
40  FOR J=1 TO N:REM -Initial Purchase of
    Chips-
50    PRINT "PLAYER ";J
60    PRINT "HOW MANY CHIPS";
70    INPUT A:A(J)=A
80  NEXT J
100 PRINT "LADIES AND GENTLEMEN! PLACE YOUR
    BETS PLEASE!"
```

```
110 FOR J=1 TO N:REM -Place Bets-
120   PRINT "PLAYER ";J
130   PRINT "NUMBER, AMOUNT";
140   INPUT B,C:B(J)=B:C(J)=C
150 NEXT J
200 LET X=INT(38*RND(0)+1):REM -Spin the
    wheel-
210 PRINT "THE WINNER IS NUMBER ";X
300 REM -Compute winnings and losses-
310 FOR J=1 TO N
340   IF X=B(J) THEN 400
350   PRINT "PLAYER ";J;" LOSES."
360   LET A(J)=A(J)-C(J):REM -Subtract amount
      lost
370   GOTO 420
400   PRINT "PLAYER ";J;" WINS ";32*C(J);
      " DOLLARS"
410   LET A(J)=A(J)+32*C(J):REM -Add
      winnings-
420 NEXT J
430 PRINT "PLAYER BANKROLLS":REM -Display
    game status-
440 PRINT
450 PRINT "PLAYER", "CHIPS"
460 FOR J=1 TO N
470   PRINT J, A(J)
480 NEXT J
500 PRINT "DO YOU WISH TO PLAY AGAIN(Y/N)";
510 INPUT R$
520 GRAPHICS 0:REM -Clears screen-
530 IF R$="Y" THEN 100:REM -Repeat game-
540 PRINT "THE CASINO IS CLOSED. SORRY!"
600 END
```

You should try a few spins of the wheel. The program is fun as well as instructive. Notice that the program lets you bet more chips than you have. In the exercises we will ask you to add in a test that checks whether there are enough chips to cover the bet. You could also build lines of credit into the game!

Notice we left a gap in the line numbering from 310 to 340. The reason is that the roulette program may be extended to incorporate the bets EVEN and ODD. If a player bets even, the player wins if the wheel stops at an even number between 2 and 36. Similarly, if a player bets odd, the player wins if the wheel stops at an odd number between 1 and 35. In either of these cases, a winning player wins the amount of his or her bet. We can easily extend the

roulette program to allow for even and odd bets. We only have to incorporate a way to make such a bet and a way to compute the winnings and losses for them. Let's handle the first problem by letting 39 stand for an odd bet and 40 an even bet. To make these choices clear let's add the line

```
90 LET ODD=39:LET EVEN=40
```

Thus player J bets odd if B(J)=ODD and player J bets even if B(J)=EVEN. Now we only have to take care of the winnings and losses for even and odd bets. To do this we will put in lines 320 and 330 (in the gap we left) to check for even and odd bets. When one is discovered, the computer will branch to line 1000 or 2000 to compute the winnings and losses, after which the computer will return to line 420 and go on to the next player. Notice that a player betting odd will lose if X=37 (corresponding to 0), if X=38 (corresponding to 00), or if X is even. A player betting even will lose if X=37, if X=38, or if X is odd. How can we test whether X is even or odd? If X is even, then X/2 is an integer, so that INT(X/2)=X/2. But if X is odd, then X/2 is not an integer, so that INT(X/2)<X/2. Now we have everything we need for the program. Here are the lines we add to the previous roulette program:

```
90   LET ODD=39:LET EVEN=40
320  IF B(J)=ODD THEN 1000
330  IF B(J)=EVEN THEN 2000
1000 IF INT(X/2)=X/2 OR X=37 OR X=38 THEN
     1500:REM -In these cases player J
     loses.-
1010 PRINT "PLAYER ";J;" WINS ";
     C(J);" DOLLARS"
1020 LET A(J)=A(J)+C(J)
1030 GOTO 420
1500 PRINT "PLAYER ";J;" LOSES ";
     C(J);" DOLLARS"
1510 LET A(J)=A(J)-C(J)
1520 GOTO 420
2000 IF INT(X/2)<X/2 OR X=37 OR X=38 THEN
     1500:REM -In these cases player J
     loses.-
2010 GOTO 1010
```

Exercises (answers on page 258)

In Exercises 1-8, write an expression involving RND that generates random numbers of the given type.

1. Numbers from 0 to 100.
2. Numbers from 100 to 101.
3. Integers from 1 to 50.
4. Integers from 4 to 80.
5. Even integers from 2 to 50.
6. Numbers from 50 to 100.
7. Integers divisible by 3 from 3 to 27.
8. Integers from among 4, 7, 10, 13, 16, 19, and 22.
9. Modify the dice program so it keeps track of payoffs and bankrolls, much like the roulette program in Example 2. Here are the payoffs on a bet of one dollar for the various bets:

OUTCOME	PAYOFF
2	35
3	17
4	11
5	8
6	6.20
7	5
8	6.20
9	8
10	11
11	17
12	35

10. Modify the roulette program of Example 2 to check that a player has enough chips to cover the bet.
11. Modify the roulette program of Example 2 to allow for a $100 line of credit for each player.
12. Construct a program which tests one-digit arithmetic facts, with the problems randomly chosen by the computer.
13. Make up a list of ten names. Write a program which will pick four of the names at random. (This is a way of impartially assigning a nasty task!)

ANSWERS TO TEST YOUR UNDERSTANDINGS 1, 2, and 3

1: 30% (or 300)

2:
```
10 FOR J=1 to 50
20   PRINT INT(2*RND(0))
30 NEXT J
40 END
```

3:
```
10 LET X=INT(3*RND(0)):REM X will be
   0, 1, or 2.
20 IF X=0 THEN PRINT "HEADS":GOTO 40
30 PRINT "TAILS"
40 END
```

5

Using Peripherals

5.1 The Cassette Recorder

In this chapter we will explain how to use the cassette recorder, the disk drive, and the printer. We will also talk about the modem.

The **cassette recorder** (also called **tape recorder** or **program recorder**) allows you to save data (such as programs) on a standard audio cassette (sometimes called a tape). See Figure 5-1 for pictures of a cassette recorder and a cassette.

At this point, we suggest you plug the cassette recorder into the computer. On the 400 and 800, plug it in on the right side of the computer. On the 600XL, 800XL, and 1200XL, plug it into the back of the computer. For more specific details, refer to the manual for the cassette recorder. Once you have the recorder plugged in your computer, plug the power cord of the cassette recorder into a power receptacle.

Since we will actually be saving and retrieving programs, we suggest you get a blank cassette and insert it into the program recorder. You can use a cassette which already has recordings on it, but the more interference (music, static, and so forth) there is on the cassette, the less chance you have of getting your data back after you save it.

Rewind the cassette and press the counter button to reset the counter to 0. This counter helps you keep track of where your program is. On most cassettes, there is a blank portion at the beginning of the tape, called the **leader,** where nothing can be recorded. Be sure to advance the tape beyond the leader to ensure that your data will be recorded. Usually, advancing the tape until the counter reaches 5 will be sufficient.

Before we can save a program, we must type one in. For purposes of illustration we will use this program:

```
10 PRINT "HELLO"
20 PRINT "NICE DAY, ISN'T IT?"
30 GOTO 10
40 END
```

Figure 5-1. A cassette recorder and a standard audio cassette.

Type in the program. After you have typed all the lines, type

```
LPRINT
```

If you do not have a printer attached, you will get an error number 138. Ignore this error message. We are about to save the program on cassette and LPRINT will increase the chances for a good save. Therefore, we recommend you always use the LPRINT command before saving a program on cassette. Now, type

```
CSAVE
```

and hit RETURN. The computer or TV should beep twice. This tells you the computer is ready to save your program. Press the

PLAY and RECORD buttons on your cassette recorder simultaneously. They should stay depressed. Now hit RETURN again. The cassette recorder should start turning and you should hear strange sounds coming from your TV set. After a little while, the cassette should stop spinning and the noise will stop. This means the computer is finished saving.

We will now try to load the program back into the computer from the cassette. Type NEW to erase the memory of your computer. Rewind the cassette until the counter returns to the value of the starting point of the program, in this case 5. Now type

```
CLOAD
```

and hit RETURN. The computer should beep at you once. This means the computer is ready to load the program from the cassette. Press PLAY on the cassette recorder and hit RETURN. The computer should be making strange noises as it did when you were saving. After a few seconds, the READY prompt should appear and the program should be in memory.

To verify this, type LIST and hit RETURN. If you got an error while loading, try typing the sample program in again, saving it, and loading it back again. The cassette recorder is not very reliable, so it is a good idea to save a program on two different places on the cassette. This will increase the chances of getting your program back.

There are other ways of saving and loading a program from cassette. These commands are used to save:

```
CSAVE
LIST "C:"
```

These commands are used to load:

```
CLOAD
ENTER "C:"
RUN "C:"
```

Programs saved with the CSAVE command can be loaded either with the CLOAD command or with the RUN "C:" command. The RUN "C:" command instructs the computer to run the program immediately after it has been loaded. To see this in operation, type NEW, rewind the cassette, and type the following:

```
RUN "C:"
```

The program should have loaded into memory and started running immediately.

We will now explain how to merge separate programs together to form one large program. Type in the following program:

```
30 PRINT
40 PRINT "THIS IS THE END OF THIS PROGRAM"
50 END
```

Next save this program on the cassette, but use the LIST "C:" command, not the CSAVE command. The LIST "C:" command is different from the LIST command you use for listing programs on the screen. It saves your program on the cassette in a way that allows you to merge it with another program. To save the program, advance the cassette counter to 20, and type the following:

```
LIST "C:"
```

Now, type NEW and load the first program into the computer. List the program to make sure you loaded the right one. Now, advance the cassette to 20 again and type

```
ENTER "C:"
```

This will merge the second program with the first. List the program and behold the finished product! Notice that line 30 of the first program was replaced by line 30 of the second program. This is because the second program has "priority" over the first program. You could now merge a third program if there were another one saved on the cassette by LIST "C:". Here are the steps for merging any two programs:

1. Type in the first program.
2. Save it on cassette.
3. Type NEW.
4. Type in the second program.
5. Save it at a different place on the cassette, using LIST "C:".
6. Load the first program into the computer.
7. Load the second program into the computer, using ENTER "C:".

Since the computer deletes any program already in memory when you CLOAD another program, you need not type NEW between steps 5 and 6 above. Remember, the ENTER "C:" statement adds a program onto the one in memory. Now you know how to store any program on cassette which you might want to keep.

5.2 The Disk Drive

The **disk drive** (sometimes called the **diskette drive**) allows you to save data (such as programs) on a **disk** (sometimes called a **diskette**). See Figure 5-2 for pictures of a disk drive and a disk.

At this point, we suggest you plug the disk drive into the computer, while everything is turned off. On the 400 and 800, plug the disk drive in on the right side of the computer. On the 600XL, 800XL, and 1200XL, plug it in on the back of the computer. For more specific details, refer to the manual for the disk drive. If you have more peripherals (such as a printer or modem) the method of connection may differ. Once you have the disk drive connected to the computer, plug the power pack of the disk drive into a power receptacle.

Since we will actually be saving programs, we suggest you have a disk handy while reading this section. Also, make sure you have an ATARI **master diskette.** Before the computer can access the disk drive, it must have the disk operating system (DOS) loaded into it. DOS has been recorded on the master diskette.

Now let's get started. Turn on the disk drive with no disk inserted. After the BUSY light goes off, insert the master diskette into the disk drive. Remember to close the door to the disk drive after you put anything in or take anything out of the disk drive. This prevents dust from collecting inside the disk drive.

Figure 5-2. A disk drive and a disk.

Turn on the computer with the BASIC cartridge inserted if you have an ATARI 400, 800, or 1200XL. The disk should spin, the BUSY light should light up, and you should hear beeping sounds coming from your TV. This means the computer is retrieving information from the disk drive. The computer is now loading DOS. After a few seconds, the READY prompt should appear. Type

```
DOS
```

and hit RETURN. The disk should spin again and a selection menu should appear on the screen. It should look something like this:

```
DISK OPERATING SYSTEM II VERSION 2S
COPYRIGHT 1980 ATARI

A. DISK DIRECTORY I. FORMAT DISK
B. RUN CARTRIDGE  J. DUPLICATE DISK
C. COPY FILE      K. BINARY SAVE
D. DELETE FILE(S) L. BINARY LOAD
E. RENAME FILE    M. RUN AT ADDRESS
F. LOCK FILE      N. CREATE MEM.SAV
G. UNLOCK FILE    O. DUPLICATE FILE
H. WRITE DOS FILES


SELECT ITEM OR RETURN FOR MENU
```

Newer or older versions may differ slightly, but it should be generally the same. We will not explain all the options in detail here, but we will tell you a little about what they are for. A disk cannot be used unless it is **formatted.** This prepares the disk for use.

> **WARNING:** Formatting a disk erases any data stored on it. Be sure it does not contain any program you wish to keep. Don't format your master diskette. It is already formatted.

To format your disk, insert it into the disk drive, type the letter I, and hit RETURN. The computer should respond with

```
WHICH DRIVE TO FORMAT?
```

At this time, hit 1. (If you have more than one disk drive, type the number of the drive that contains the disk to be formatted.) This tells the computer you want it to format the disk in drive one.

Once you hit RETURN, the computer should display something like

```
TYPE "Y" TO FORMAT DISK 1
```

This is a precautionary step to avoid accidental formatting and to allow you to change your mind about formatting a disk. If you type N and hit RETURN, the formatting process will be aborted. If you type Y and hit RETURN, your disk will start spinning and the computer will format your disk. After it is finished formatting the disk, the computer will print

```
SELECT ITEM OR RETURN FOR MENU
```

on the screen. Now the disk is ready for you to write information on it.

The first information you might like to put on your disk is a copy of DOS. The advantage of having DOS on the disk is that you will not have to use the master diskette every time you want to use your disk. Whenever you put your disk into the drive and turn on the computer, DOS will automatically load into the computer's memory. Now, hit RETURN. The screen will clear and the DOS menu will be printed on the screen. To write DOS files on your disk, type H and hit RETURN. The computer should respond:

```
DRIVE TO WRITE DOS FILES TO?
```

When you type 1, the disk will spin and the computer will write DOS on your disk. Now hit RETURN to get the DOS menu back on the screen.

To verify that DOS has been put on your disk, type A and hit RETURN twice. This prints out a directory of all the programs on the disk. It also will print out how long each program is in sectors. A **sector** is a part of the disk. Each sector can hold a set amount of information. At the bottom of the list of programs, the computer will tell you how many free sectors are left on the disk. After you have finished looking at the list, hit RETURN. You should see the DOS menu printed out again. Type B for RUN CARTRIDGE and hit RETURN. You should now be back in BASIC, and you should see the READY prompt.

You are now ready to write a program and save it on the disk. For purposes of illustration, we will use this program:

```
10 PRINT "HELLO"
20 PRINT "NICE DAY, ISN'T IT?"
30 END
```

Type in the program. Then type

```
SAVE "D1:HELLO"
```

and hit RETURN. The disk will spin and you should hear beeping sounds from your TV set or monitor. The above command tells the computer you want to save your program on the disk in drive #1 under the name HELLO. If you want to save or load a program from drive 1, you need not type D1, merely D. This is a shortcut which is commonly used. You could have substituted other names for HELLO, but you must follow certain rules:

1. The name can be up to 8 characters long, plus an optional three-character extender, which you separate from the main part of the name with a period (.). Here are some examples:

   ```
   HELLO
   QD1
   A1B2
   TELETYPE.DAT
   TELETYPE.BAS
   ```

2. The name must follow the same rules as a variable. To see these rules, review Section 2.4.

We will now try to load the program back into the computer from the disk. Type NEW to delete the program you have in memory. Type LIST to check if there is a program in memory or not. Now type

```
LOAD "D:HELLO"
```

and hit RETURN. You should hear beeping noises as you did before and almost immediately the program will be loaded into the computer from the disk. Try typing LIST to verify this.

There are other ways of saving and loading a program from disk. These commands are used to save:

```
SAVE "D:<FILENAME>.<EXTENDER>"
LIST "D:<FILENAME>.<EXTENDER>"
```

These commands are used to load:

```
LOAD "D:<FILENAME>.<EXTENDER>"
ENTER "D:<FILENAME>.<EXTENDER>"
RUN "D:<FILENAME>.<EXTENDER>"
```

Programs saved on disk with the SAVE command can be loaded either with the LOAD command or the RUN command. The RUN command used with the disk drive instructs the computer to run a program immediately after loading it. To see this in operation, type NEW and then type the following:

```
RUN "D1:HELLO"
```

The HELLO program should have loaded into memory, and it should have started running immediately.

We will now explain how to merge separate programs together to form one large program. Type the following program in:

```
30 PRINT
40 PRINT "THIS IS THE END OF THIS PROGRAM"
50 END
```

Save this program on the disk, using the LIST command with the form LIST "D:<FILENAME>.<EXTENDER>". This command is different from the LIST command you use for listing programs on the screen. It will save your program on the disk in a way which allows you to merge it with another program. Here is what you should type:

```
LIST "D:HELLO2"
```

Type NEW and load the first program (HELLO) in with the command

```
LOAD "D:HELLO"
```

Type LIST to make sure you loaded the right program. Now, type

```
ENTER "D:HELLO2"
```

This will merge the second program with the first. List the program and behold the finished product! Notice line 30 of the first program was replaced by line 30 of the second program. This is because the second program has "priority" over the first program. We could now merge a third program if there were another one stored on the disk. Here are the steps for merging any two programs:

1. Type in the first program.
2. Save it on disk.
3. Type NEW.
4. Type in the second program.
5. Save the program on disk using the command LIST "D:<FILENAME>.<EXTENDER>. Be sure to save it with a different filename than the first program.
6. Load the first program into the computer.
7. Load the second program into the computer using the ENTER command.

Since the computer deletes any program already in memory when you load another program, you need not type NEW between steps

5 and 6 above. Remember, the ENTER command adds a program onto the one in memory.

Now you know how to store on diskette any program which you might like to keep. Always remember to start your system by turning on the disk drive, inserting a disk with DOS on it, and then turning on the computer with the BASIC cartridge inserted.

5.3 The Printer

We will not go into much detail about printers because there are so many that can be used with ATARI computers. Many non-ATARI printers require the ATARI 850 INTERFACE MODULE to be used with ATARI computers. The method of connecting the components of your system together depends upon what hardware you own and what type of printer you have. If the manual doesn't explain how to connect the computer system together, just connect it any way it will fit, and it will probably work out. After you have your system connected, make sure you have all the components plugged into a wall socket. When starting up the computer system, remember to turn on the computer last.

You must use special commands to send information to the printer. To have any kind of text printed by your printer, use the LPRINT command. To see the effect of this command, type the following:

```
LPRINT "HELLO"
```

The printer should have printed HELLO on the printer paper. You can use the LPRINT command in programs as well as in the direct mode.

TEST YOUR UNDERSTANDING 1 (answer on page 127)

Write a program to have the printer print the following:

Dear COMPUTER INFORMATION magazine:

I am interested in having a year's subscription of your magazine. I understand the cost is $15.95 per year. My check is enclosed.

Sincerely,
Mark Ellis

Often you want a printed copy (or hardcopy) of a program. If you want to list a program that is in memory onto the printer, type

```
LIST "P:"
```

and hit RETURN. Try listing this program onto the printer using LIST "P:":

```
10 PRINT "HELLO"
20 FOR A=1 TO 5
30   PRINT "NICE DAY...ISN'T IT???"
40 NEXT A
50 END
```

Suppose you have a program that prints something on the screen, but you would like to have it printed on the printer? If you type

```
POKE 838,166: POKE 839,238
```

any text that would normally be printed on the screen will be printed on the printer. To "undo" this, simply type

```
POKE 838,244: POKE 839,241
```

Now, type in this program:

```
10 PRINT "LINE 10"
20 FOR A=1 TO 10
30   PRINT "LINE 30"
40 NEXT A
50 PRINT "LINE 50"
60 END
```

Run it and watch the screen. Now, add these lines to the program:

```
5  POKE 838,166: POKE 839,238
55 POKE 838,244: POKE 839,241
```

Run the program again, but this time watch the printer. Line 5 instructs the computer to send to the printer everything that would normally be printed on the screen. Line 55 returns everything to normal.

ANSWER TO TEST YOUR UNDERSTANDING 1

1:
```
10 LPRINT "Dear COMPUTER INFORMATION
   magazine:"
20 LPRINT
30 LPRINT " I am interested in having
   a year's subscription"
40 LPRINT "of your magazine. I
   understand the cost is $15.95"
```

```
50 LPRINT "per year. My check is
   enclosed."
60 LPRINT
70 LPRINT "                    Sincerely,"
80 LPRINT "                    Mark Ellis"
90 END
```

5.4 The Modem

A **modem** (short for **mo**dulator/**dem**odulator) makes it possible for one computer to talk to another computer. The ATARI 830 modem can be used only if you have an ATARI 850 INTERFACE MODULE. Many other modems are available for ATARI computers.

Modems send special computer code over telephone lines. The computer on the other end of the line can understand these codes, and thus the computers "talk" to each other. Modems can be used to swap programs, get stock market reports, or do anything having to do with information exchange. You could even have a conversation with another person not by talking over the telephone line, but by typing information on the keyboard. In fact, special **bulletin boards** are set up so people can talk to each other, leave messages for each other, swap programs, and so forth.

Modems can be difficult to program, so we suggest you buy a telecommunications program for use with your modem. Such a program is available in any good computer store. The manual with the program should have proper instructions on how to use the modem with the program.

6

Computer Graphics and Text

6.1 Introduction

So far we have displayed only normal, small-sized text on the screen. However, it is possible to display text in large letters and even to draw pictures on the screen. In this chapter, we will learn how to create such effects by using the various display modes available on ATARI computers.

A **display mode** is any mode you can enter that lets you somehow put something on the screen (such as text or pictures). You are already familiar with one display mode, the one in effect when the computer is first turned on, and in which you have small light lettering on a blue background. In this chapter, we will explain the other display modes. You will learn how to display many sizes of text in various colors using certain display modes called **text modes.**

When you draw pictures, graphs, etc. on the screen, you are using **graphics.** ATARI computers have some of the best graphics capabilities of all home computers. You create graphics by causing the computer to enter a graphics mode. There are several **graphics modes.** Each one has its own advantages and disadvantages. Some have more colors than others, some are capable of greater detail than others, and some use less memory than others. We will not discuss memory because you probably will not get into programs which will "overflow" the memory.

Many ATARI computers have a special graphics chip called the GTIA. It allows three additional graphics modes. All ATARI XL computers and most 400s and 800s sold after January, 1982, have the GTIA chip. We will show you how to find out whether your ATARI has a GTIA chip when we discuss the special graphics modes. Here is a list of all the display modes on the various computers:

ATARI 400 or 800 without GTIA
Graphics modes 3, 4, 5, 6, 7, 8
Text modes 0, 1, 2

ATARI 400 or 800 with GTIA
Graphics modes 3, 4, 5, 6, 7, 8, 9, 10, 11
Text modes 0, 1, 2

ATARI 600XL, 800XL, and 1200XL
Graphics modes 3, 4, 5, 6, 7, 8, 9, 10, 11, 14, 15
Text modes 0, 1, 2, 12, 13

The Screen

Turn your computer on and observe the screen. Notice there is a blue **background** surrounded by a black **border.** (On some TV sets it may be difficult to distinguish the border from the background.) The READY prompt appears in light blue on top of the background. It is part of the **foreground**, as would be any text or graphics. Any picture that you draw on the screen in a graphics mode would also be part of the foreground. The border, background, and foreground are different colors, which makes it possible for us to tell them apart. As we will explain later, it is possible in most modes to control the color of the border, background, and foreground in order to create various effects. In some modes, the border must be the same color as the background. In effect, the border has been eliminated in these modes.

Foreground display (text or graphics) can be placed anywhere on the background. In all graphics modes, the background is divided into small rectangles, each called a **pixel** ("picture element"). In mode 8, for example, there are 320 columns and 160 rows of pixels. The columns are numbered from 0 to 319 and the rows are numbered from 0 to 159. The other graphics modes have different numbers of columns and rows.

In all text modes, each letter, number, or symbol on the screen is called a **character**. In mode 0, for example, there are 40 columns and 24 rows of characters. The columns are numbered from 0 to 39 and the rows are numbered from 0 to 23 (see Figure 6-1). The other modes have different numbers of columns and rows.

Let us explain the method for referring to pixel and character locations on the screen. In each mode, the columns are numbered from left to right, starting with 0 on the left. The rows are numbered from top to bottom, starting with 0 at the top. (The numbering system for mode 0 is shown in Figure 6-1.) Since each pixel or character lies in a column and a row, we can specify its location by giving its column and row number.

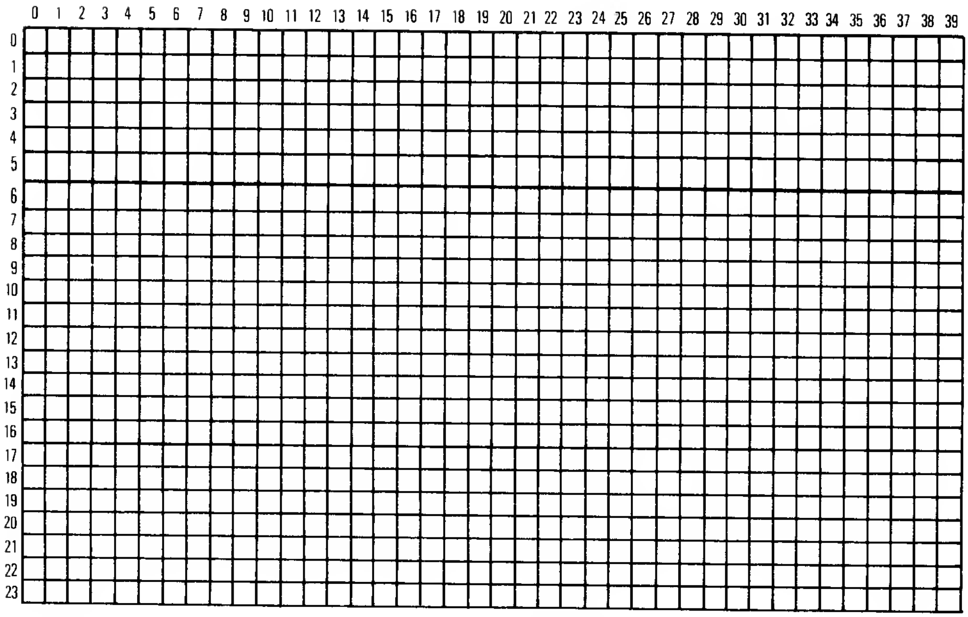

Figure 6-1. A grid illustrating the resolution of graphics mode 0.

The column number is usually called the **X coordinate,** and the row number is usually called the **Y coordinate.** Together the column and row numbers are called the **coordinates** of the pixel or character. The X coordinate (column number) is always given first. For example, in mode 0 the character with coordinates 17,5 lies in column number 17 and row number 5 (see Figure 6-2). However, remember that the numbering of columns and rows begins with 0, so column number 17 is actually the eighteenth column from the left and row number 5 is the sixth row from the top. The more columns or rows there are, the smaller the pixels or characters are. The numbers of columns and rows determine the **resolution** of the mode. For example, the resolution of mode 0 is 40×24.

6.2 Graphics Commands

Before we discuss the individual display modes, we will explain the commands which control the display on the screen. The most basic command is PRINT, which you already know. Other commands choose the display mode, control color, draw lines on the

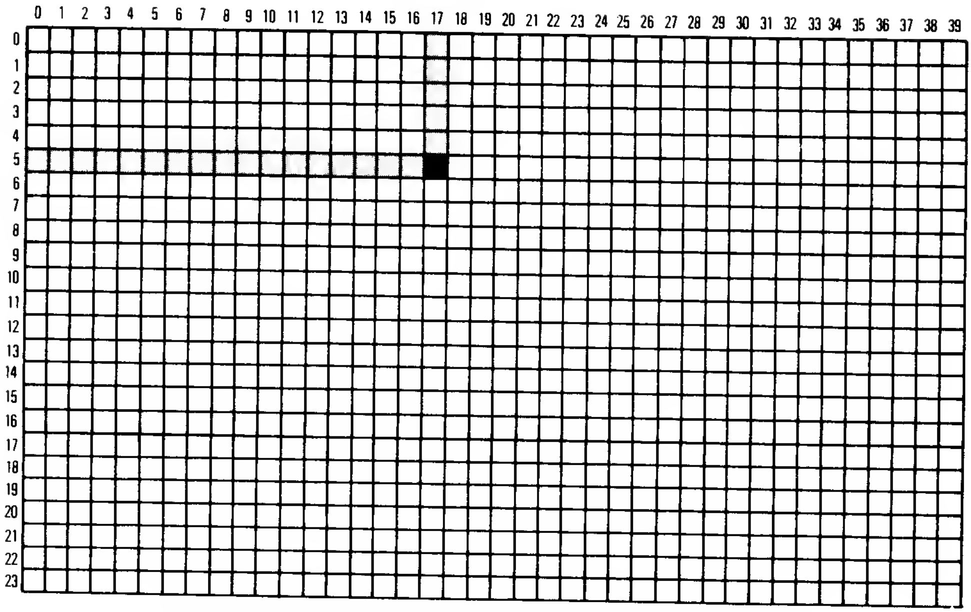

Figure 6-2. The coordinates of a character in mode 0.

screen, or serve some other purpose related to display. Let's look at the commands one at a time.

GRAPHICS

In order to enter into any of the display modes, including text modes, you must use the command

```
GRAPHICS X
```

where X is the number of the desired mode. The command may be given in the direct mode or in a program. For example, the command

```
10 GRAPHICS 8
```

at the beginning of a program will cause the computer to enter into display mode 8. You can then continue the program and instruct the computer to draw a specified design in the display mode. "GR." is an abbreviation for "GRAPHICS". Thus, the following two statements have the same meaning:

```
GRAPHICS 8
GR. 8
```

When the computer enters into a graphics mode, the screen will clear. Then, in all modes except 9 through 11, a portion of the screen four lines high will appear at the bottom of the screen where text can be written. This is called the **text window.** Try typing this:

```
GRAPHICS 3
```

The screen should have turned black and there should be a blue text window at the bottom of the screen. The READY prompt will be visible in the text window. You can enter commands in the direct mode whenever the text window is present. To get rid of the text window at the bottom of the screen in any graphics mode (except 9-11) to have more room for pictures, just add 16 to the graphics mode number. For example, to eliminate the text window from GRAPHICS 3, give the command

```
GRAPHICS 3+16
```

or

```
GRAPHICS 19
```

If, while in the direct mode, you try to enter into graphics modes 9-11 or any graphics mode without a text window, the screen will flash and return to mode 0. This happens because these modes do not have text windows. The computer tries to display READY on the screen, but because there is no place to write it, the computer must go back to mode 0. So, if you want to use a mode without a text window, do it in a program.

Mode 0 is the default mode; that is, the one that is in effect when you first turn on the computer. When you enter into a different mode, you will no longer be in mode 0, and thus all the text and information on the screen will be erased. Programs will stay in memory no matter what mode you are in. It is not necessary to give a GRAPHICS command if you wish to use only mode 0 because it is default. However, the command GRAPHICS 0 automatically clears the screen, so you can use this command in a program to have the screen cleared.

SETCOLOR

The SETCOLOR command has the format

```
SETCOLOR A,B,C
```

where A, B, and C are integers. The number A must be one of the numbers 0, 1, 2, 3, or 4. These five numbers refer to five **color registers** in the memory. The color register determined by the number A records the numbers B and C, which (in most cases) refer to **hue** (color) and **luminance** (brightness of the color). The use of the various registers varies somewhat from mode to mode, so we will explain it in detail with each mode. Basically, in each mode, the number A specifies what is to be colored. For example,

you might wish to specify the color of the background of the screen in mode 0. The value of A for this is 2. That is, in mode 0 the hue and luminance stored in color register #2 will determine the hue and luminance of the background. If you type in

```
SETCOLOR 2,12,4
```

the background will turn to a shade of green. (The exact shade will vary from TV set to TV set.) If you replace 2 with 4, the border of the screen will turn green because color register #4 controls the hue and luminance of the border in GRAPHICS 0:

```
SETCOLOR 4,12,4
```

The number B represents hue and can be any integer from 0 to 15. To see the various hues available, type in and run this program, which turns the background every possible hue (but not every possible luminance of the hue), one at a time:

```
10 FOR B=0 TO 15
20   PRINT "COLOR ";B
30   SETCOLOR 2,B,6
40   FOR D=1 TO 500:REM -DELAY-
50   NEXT D
60 NEXT B
```

The number C represents brightness (luminance) and can be any even integer from 0 to 14. The larger C is, the brighter the color. To see how C controls luminance, alter the preceding program so it becomes

```
10 FOR B=0 TO 15
20   PRINT "COLOR ";B
25   FOR C=0 TO 14 STEP 2
26     PRINT "LUMINANCE ";C
30     SETCOLOR 2, B, C
40     FOR D=1 TO 250
50     NEXT D
55   NEXT C
60 NEXT B
```

Essentially, B selects the color, and C selects the shade of that color. Table 6-1 lists the hues each value of B refers to (the color may vary for your TV set).

There are default values of B and C for each register. This means that if you do not give a SETCOLOR command, the computer will use the default values of B and C in all the color regis-

TABLE 6-1. SETCOLOR numbers.

0 gray	8 medium blue
1 light orange or gold	9 light blue
2 orange	10 turquoise
3 red-orange	11 green-blue
4 pink	12 green
5 lavender	13 yellow
6 blue-purple	14 orange-green
7 purple-blue	15 light orange

ters. The following table lists the default values and the actual hues corresponding to B. They are the same for all modes.

A	B	C	COLOR
0	2	8	orange
1	12	10	light green
2	9	4	dark blue
3	4	6	pink
4	0	0	black

COLOR

The COLOR command, which is different from SETCOLOR, is used to select a color from one of the color registers. Its form is

```
COLOR X
```

It is useless unless used in conjunction with other commands. For example, suppose you wanted to draw a line on the screen. (Don't worry about how to do it. We will explain that shortly.) Before giving the command for drawing the line, you would first specify the hue and luminance of the line by giving a COLOR command. If you gave the command

```
COLOR 2
```

the line would be drawn with the hue and luminance stored in the register that controls COLOR 2. You will become more familiar with the COLOR statement as we introduce more commands, but for now let us try to point out the difference between SETCOLOR and COLOR.

The SETCOLOR command is used to store hues and luminances in the color registers. It actually changes the values in the color reg-

isters. It can change the hue and/or luminance of the background, border, or foreground (a picture, for example) of the screen. Although you have a wide choice of which hues and luminances to store, only those that have been stored (by you or by default) can actually be used at any particular time. There is a definite limit to the number of colors you can use in a picture or diagram.

In contrast, the COLOR command is used to select from among the hues and luminances stored in the registers. It can't change the color of anything already on the screen. Instead, it selects a color register whose hue and luminance are to be used with subsequent PLOT and DRAWTO statements, which we will explain next.

CAUTION: COLOR 1 may not pertain to SETCOLOR 1. The same applies to the other COLOR numbers.

PLOT

The PLOT command is used in graphics modes to "plot a point" on the screen. We do this by coloring a pixel. We have to tell the computer which color to use and which pixel to color. The color is selected by using a COLOR statement, and the pixel is then specified by a PLOT statement. The form of a PLOT statement is

```
PLOT X,Y
```

where X and Y are the coordinates of the pixel (see Section 6.1).

Now let us put everything together and actually plot a point on the screen. Let's use mode 3. According to Table 6-2, mode 3 is a four-color mode with a text window and a resolution of 40×20. COLOR 1 selects the hue and luminance stored by SETCOLOR 0, whose default color is a shade of orange (2,8).

Let's plot the point with coordinates 20,10 (near the center of the screen) in the default color. Here is the program:

```
10 GRAPHICS 3:REM -Enter mode 3-
20 COLOR 1:REM -Select color (hue and
   luminance) of register 0-
30 PLOT 20,10:REM -Plot point in selected
   color-
```

Now suppose we wanted to plot a pink point with medium luminance at 20,10. According to Table 6-1, the number of pink is 4. Let's use 8 for the luminance. We first store the hue and luminance using SETCOLOR 0 and then plot as before:

```
10 GRAPHICS 3:REM -Enter mode-
20 SETCOLOR 0,4,8:REM -Store pink in register
   0-
30 COLOR 1:REM -Select color stored in
   register 0-
40 PLOT 20,10:REM -Plot point in the selected
   color-
```

You could change the hue or luminance of the pixel by changing the second and third numbers in line 20. You could change where the point is plotted by changing the numbers in line 40. The following program plots points at random places on the screen with random hues and luminances:

```
10  GRAPHICS 3
20  LET B=INT(16*RND(0))
30  LET C=2*INT(8*RND(0))
40  SETCOLOR 0,B,C
50  COLOR 1
60  LET X=INT(40*RND(0))
70  LET Y=INT(20*RND(0))
80  PLOT X,Y
90  FOR DELAY=1 TO 200:NEXT DELAY
100 GOTO 10
```

Notice that at line 40 the program stores a random hue and luminance in register zero, and at line 80 plots a point at a random location with the stored hue and luminance. Also notice that line 100 sends the computer back to line 10, where it encounters the command GRAPHICS 3. This command automatically clears the screen, so the point that was just plotted is erased. Then a new hue and luminance are stored and another point is plotted at a random location with the newly stored hue and luminance, and so forth. Now change the program so line 100 becomes

```
100 GOTO 20
```

Now the computer no longer encounters the GRAPHICS 3 command after plotting each point, so the screen is not cleared. This means that once a point is plotted, it stays on the screen, and the number of plotted points gradually increases.

When you run the new program, you will notice that each time a new point is plotted, all the plotted points will change to the new color. This is due to the fact that the hue and luminance of anything plotted with COLOR 1 is controlled by a single color register. As soon as the hue or luminance stored in that register is changed, the computer automatically changes everything plotted

Table 6–2. Graphics modes.

Mode	Resolution	# of Colors	SETCOLOR #	Use	Default value
3	40×20 w/window 40×24 w/o window	4	0	Color 1	Orange (2,8)
			1	Color 2, luminance of text in text window	Light Green (12,10)
			2	Color 3, color of text window	Dark Blue (9,4)
			3	NO USE	Pink (6,4)
			4	Color 4, background color, border color	Black (0,0)
4	80×40 w/window 80×48 w/o window	2	0	Color 1	Orange (2,8)
			1	Luminance of text in text window	Light Green (12,10)
			2	Color of text window	Dark Blue (9,4)
			3	NO USE	Pink (4,6)
			4	Color 2, background color, border color	Black (0,0)
5	80×40 w/ window 80×48 w/o window	4	0	Color 1	Orange (2,8)
			1	Color 2, luminance of text in text window	Light Green (12,10)
			2	Color 3, color of text window	Dark Blue (9,4)
			3	NO USE	Pink (6,4)
			4	Color 4, background color, border color	Black (0,0)
6	160×80 w/window 160×96 w/o window	2	0	Color 1	Orange (2,8)
			1	Luminance of text in text window	Light Green (12,10)
			2	Color of text window	Dark Blue (9,4)
			3	NO USE	Pink (4,6)
			4	Color 2, background color, border color	Black (0,0)
7	160×80 w/window 160×96 w/o window	4	0	Color 1	Orange (2,8)
			1	Color 2, luminance of text in text window	Light Green (12,10)
			2	Color 3, color of text window	Dark Blue (9,4)
			3	NO USE	Pink (6,4)
			4	Color 4, background color, border color	Black (0,0)
8	320×160 w/window 320×192 w/o window	1 color 1 luminance	0	NO USE	Orange (2,8)
			1	Color 1, luminance of text in text window	Light Green (12,10)
			2	Background color, color of text window	Dark Blue (9,4)
			3	NO USE	Pink (4,6)
			4	Border color	Black (0,0)

Table 6–2. Graphics modes (continued).

Mode	*Resolution*	*# of colors*	*SETCOLOR #*	*Use*	*Default value*
9	80×192	1 hue	0	NO USE	Orange (2,8)
		16	1	NO USE	Light Green (12,10)
		luminances	2	NO USE	Dark Blue (9,4)
			3	NO USE	Pink (6,4)
			4	Background color, border color + special	Black (0,0)
10	80×192	9	0	Color 4	Orange (2,8)
			1	Color 5	Light Green (12,10)
			2	Color 6	Dark Blue (9,4)
			3	Color 7	Pink (6,4)
			4	Color 8	Black (0,0)
11	80×192	16 hues	0	NO USE	Orange (2,8)
		1 luminance	1	NO USE	Light Green (12,10)
			2	NO USE	Dark Blue (9,4)
			3	NO USE	Pink (6,4)
			4	Color (but not luminance) of background, border color, luminance of all colors	Black (0,0)
14	160×160 w/window	2	0	Color 1	Orange (2,8)
	160×192 w/o window		1	Luminance of text in text window	Light Green (12,10)
			2	Color of text window	Dark Blue (9,4)
			3	NO USE	Pink (6,4)
			4	Color 2, background color, border color	Black (0,0)
15	160×160 w/window	4	0	Color 1	Orange (2,8)
	160×192 w/o window		1	Color 2, luminance of text in text window	Light Green (12,10)
			2	Color 3, color of text window	Dark Blue (9,4)
			3	NO USE	Pink (6,4)
			4	Color 4, background color, border color	Black (0,0)

with COLOR 1. The only way to get different colors on the screen at the same time is to use different color registers.

As we said, GRAPHICS 3 is a four-color mode. This is due to the fact that there are four color registers available to you in that mode. Now refer to Table 6-2 again. You see, for example, that the color register corresponding to SETCOLOR 4 controls COLOR 4, the background, and the border. Therefore, the background, the border, and anything plotted with COLOR 4 will all have the same hue and luminance. Three other color registers can be set using SETCOLOR with 0, 1, or 2, and these control COLOR 1, COLOR 2, and COLOR 3, respectively. Notice that SETCOLOR 3 has no use in mode 3.

Now let's write a program which plots in four different colors. Let's use the default colors in the registers so we won't have to use SETCOLOR. We will choose COLOR 1, 2, 3, or 4 at random and plot a point at a random location:

```
10 GRAPHICS 3
20 LET C=INT(4*RND(0))
30 COLOR C
40 LET X=INT(40*RND(0))
50 LET Y=INT(20*RND(0))
60 PLOT X,Y
70 FOR D=1 TO 200:NEXT D
80 GOTO 20
```

When you run this program, you will sense only three colors plotting on the screen. The reason is that one of the plotting colors is the color of the background, so a point plotted in that color cannot be distinguished from the background. Occasionally you will see a point change color or seem to "disappear." This occurs when a point is replotted with a new hue or luminance. Whenever a point is replotted with the background color, it seems to disappear.

If you use a graphics mode other than mode 3, the same ideas apply, but the use of the color registers, the number of colors available, the resolution, etc. may be different. You will find it convenient to refer to Table 6-2 when using an unfamiliar mode. With it, you should be able to use each mode effectively.

DRAWTO

The DRAWTO statement is used to draw a line on the screen. There are three things you need to tell the computer (assuming a graphics mode has already been selected): the color of the line, the starting point of the line, and the finishing point of the line. Of

course, the color is specified by a COLOR statement. The starting point is specified by a PLOT statement, and the finishing point by a DRAWTO statement. The form of the DRAWTO statement is

```
DRAWTO X,Y
```

where X and Y are the coordinates of the finishing point of the line. For example, the following program draws a red-orange (3,8) line in mode 3 from the point 0,0 to the point 39,19:

```
10 GRAPHICS 3
20 SETCOLOR 0,3,8
30 COLOR 1
40 PLOT 0,0
50 DRAWTO 39,19
```

Because mode 3 has such large pixels (low resolution), diagonal lines do not appear to be very smooth.

If you wish to draw several lines in succession, each starting where the previous one ends, you need only one PLOT statement (for the first point). Here is an example:

```
10 GRAPHICS 8
20 SETCOLOR 2,0,0
30 COLOR 1
40 PLOT 0,0
50 DRAWTO 100,0
60 DRAWTO 100,100
70 DRAWTO 0,100
80 DRAWTO 0,0
90 END
```

DRAWTO may be abbreviated to DR. (The period is necessary.) For example, the following statements are equivalent:

```
DRAWTO 0,0
```

and

```
DR. 0,0
```

PRINT

When you use a graphics mode with a text window, you can use the PRINT statement in the usual way to print in the text window. The following program is an illustration. It draws the "same" figure on the screen in each of the modes 3-8. Because of the different resolutions of the various modes, the size of the figure will vary. In the text window the program will print the mode currently in use.

```
10  FOR M=3 TO 8
20    GRAPHICS M
30    IF M=8 THEN SETCOLOR 2,0,0:REM -Turn
      screen black in mode 8-
40    COLOR 1
50    PRINT "MODE ";M
60    FOR H=0 TO 16 STEP 4
70      PLOT 0,H
80      DRAWTO 16,H
90    NEXT H
100   FOR V=0 TO 16 STEP 4
110     PLOT V,0
120     DRAWTO V,16
130   NEXT V
140   FOR D=1 TO 1000
150   NEXT D
160 NEXT M
```

Remember the text window can be eliminated by adding 16 to the mode number. In that case, the part of the screen available for graphics is increased by several rows of pixels. For example, mode 3 is 40×20, whereas mode 3+16 (mode 19) is 40×24. Try changing some of the plotting programs so they will plot on the entire 40×20 background in mode 3+16. If you use a PRINT statement in a graphics mode without a text window, the computer will return to mode 0, clear the screen, and print whatever is called for by the PRINT statement.

6.3 Graphics Modes

In this section we will discuss all the graphics modes, one by one. Table 6-2 on page 138 contains all the pertinent information about the various graphics modes you will need for this section.

Modes 3-7

These modes all have text windows and use the same color for the border and the background. They fall into two general groups, two- and four-color modes. Modes 4 and 6 are two-color modes, with COLOR 2 being the color of the background and border, while modes 3, 5, and 7 are four-color modes, with COLOR 4 being the color of the background and border.

The advantage of modes 4 and 6 is that they conserve memory. Since most of your applications or needs will probably not exceed

the memory of your computer, it might be better to use only the 4 color modes, modes 3, 5, and 7. The resolutions of these three modes are 40×20, 80×40, and 160×80, respectively. Therefore, the area of each pixel in mode 3 is four times the area of each pixel in mode 5, which in turn is four times the area of each pixel in mode 7. To see the effect this changing resolution has, type in and run the following program, which draws two diagonal lines in each of these modes without a text window:

```
10  FOR N=0 TO 2
30    GRAPHICS 3+2*N+16:REM -Mode 3, 5, or 7
      without text window-
30    COLOR 1
40    PLOT 0,0
50    DRAWTO 40*2^N-1, 20*2^N-1:REM -The
      coordinates of lower right corner-
60    COLOR 3
70    PLOT 0, 20*2^N-1:REM -The coordinates
      of the lower left corner-
80    DRAWTO 40*2^N-1, 0:REM -The coordinates
      of the upper right corner-
90    FOR D=1 TO 2000:NEXT D
100 NEXT N
```

Notice there are no SETCOLOR statements in the program, so we are using default colors and luminances. One diagonal is drawn in COLOR 1, which is orange, and the other in COLOR 3, which is dark blue. The background (COLOR 4) is black.

TEST YOUR UNDERSTANDING 1 (answer on page 149)

Alter the the last program so that the first diagonal is drawn in light green (12,10) and the other in gold (1,6).

Mode 8

This mode is quite different from modes 3-7. It has two colors and very good resolution, so it is good for detailed picures that do not need more than two colors. The background and border may be different colors. The background hue and luminance are controlled by color register 2, and the hue and luminance for the border are controlled by color register 4. COLOR 2 has the hue and luminance of the background, so it can be used to "erase" from

the screen. COLOR 1 uses the same hue as the background, but the luminance from color register 1. The hue in color register 1 plays no role in mode 8. Therefore, to set the luminance for COLOR 1, you can use the command

```
SETCOLOR 1,X,Y
```

where Y is the desired luminance number and X can be any number.

We will illustrate mode 8 with the following program which generates a "random walk." We start by plotting a point in the center of the screen. Then, at random, we will draw a line (walk) in one of four directions: up, down, right, or left. This step will be repeated until the program is stopped. The direction of the walk will be determined by choosing one of the numbers 1(up), 2(down), 3(right), or 4(left) at random. We will use A,B for the coordinates of the moving point. The length of each move will be 10 pixels. To move up, we decrease B by 10. To move down, we increase B by 10. To move right or left, we increase or decrease A by 10. Of course, we will have to check to be sure the line doesn't go off the screen. Here is the program:

```
10  GRAPHICS 8+16
20  SETCOLOR 2,0,0:REM -Turns the background
    black-
30  COLOR 1
40  LET A=180:LET B=98:REM -Coordinates of
    starting point-
50  PLOT A,B
60  LET D=INT(4*RND(0))+1: REM -Choose 1,2,3,
    or 4 at random-
70  IF D=1 AND B>9 THEN LET B=B-10
80  IF D=2 AND B<182 THEN LET B=B+10
90  IF D=3 AND A<310 THEN LET A=A+10
100 IF D=4 AND A>9 THEN LET A=A-10
110 DRAWTO A,B
120 GOTO 60
130 END
```

When you run this program, you will observe that the horizontal lines are drawn in one color, and the vertical lines are drawn in a different color, even though there is only one COLOR statement in the program. The "fake coloring" of lines in this way is called **artifacting**. It occurs in mode 8 only and is caused by the way the TV handles color. There are specific rules (which we will not discuss) about how to achieve certain colors. If you know how to use artifacting to your advantage, you can create fantastic multi-color pictures such as this:

```
10  GRAPHICS 8+16:SETCOLOR 2,0,0:COLOR 1
20  FOR A=0 TO 40 STEP 2
30    PLOT A,0:DRAWTO A,191
40  NEXT A
50  FOR A=21 TO 61 STEP 2
60    PLOT A,0:DRAWTO A,191
70  NEXT A
80  FOR A=0 TO 80 STEP 2
90    PLOT 80,A:DRAWTO 160,A+40
95    PLOT 200,A:DRAWTO 280,A+40
100 NEXT A
110 FOR A=20 TO 120 STEP 2
120   PLOT 101,A:DRAWTO 181,A+40
125   PLOT 221,A:DRAWTO 301,A+40
130 NEXT A
140 FOR A=80 TO 182 STEP 2
150   PLOT A,0:DRAWTO A,191
160 NEXT A
900 GOTO 900
999 END
```

Here is another program using mode 8:

```
10  GRAPHICS 8+16
20  SETCOLOR 2,0,0
30  COLOR 1
40  FOR A=0 TO 191 STEP 5
50    B=A*1.67
60    PLOT 0,A
70    DRAWTO B,191
80    DRAWTO 319,191-A
90    DRAWTO 319-B,0
100   DRAWTO 0,A
110 NEXT A
120 SETCOLOR 2,0,14
130 SETCOLOR 1,0,0
140 GOTO 140
150 END
```

Try to make your own high-resolution pictures using mode 8. You'll be surprised at what you can accomplish.

Modes 9-11

These modes can be used only on computers with the GTIA chip. All ATARI XL computers have this chip, and most of the 400s and 800s sold after January, 1982, have it. Here is a program to test whether you have the GTIA chip:

```
10 GRAPHICS 9
20 FOR C=0 TO 16
30   FOR L=0 TO 16
40     SETCOLOR 4,A,B
50   NEXT L
60 NEXT C
70 GOTO 20
80 END
```

When you run this program, colors will flash on the screen. If you have the GTIA chip, the border will always be the same color as the background. If you do not have the GTIA chip, the border will have a different color from the background and hence it will be clearly visible.

Modes 9-11 are perhaps the most unusual modes on the ATARI computers. You can create multi-colored or shaded pictures with these modes. All of these modes have no text window, and all have 80×192 resolution. Mode 9 has one hue and sixteen luminances (or brightnesses of the hue). You control the color of the background, border, and COLOR 0 with SETCOLOR 4. Each of the 16 luminances is a "shade" of the hue of the background. Here is a program illustrating the effect of shading in mode 9:

```
10  GRAPHICS 9
20  FOR C=0 TO 15
30    COLOR C
40    PLOT C+25,0:DRAWTO C+25,191
50    PLOT 56-C,0:DRAWTO 56-C,191
60  NEXT C
70  FOR A=0 TO 15
80    SETCOLOR 4,A,0
90    FOR D=1 TO 500
100   NEXT D
110 NEXT A
120 END
```

Here is an even better display of the 3-D shading capabilities of mode 9:

```
10  GRAPHICS 9
20  LET S=0:LET X=68:LET C=1
30  FOR A=S TO S+15
40    COLOR C
50    LET C=C+1
60    PLOT 0,A:DRAWTO X,A
70  NEXT A
80  LET C=15
90  FOR A=S+15 TO S+30
```

```
100    COLOR C
110    LET C=C-1
120    PLOT 0,A:DRAWTO X,A
130 NEXT A
140 LET X=X-28:LET C=1
150 FOR A=X+1 TO X+15
160    COLOR C
170    LET C=C+1
180    PLOT A,30-C:DRAWTO A,191
190 NEXT A
200 LET C=15
210 FOR A=X+16 TO X+29
220    COLOR C
230    C=C-1
240    PLOT A,C-1:DRAWTO A,191
250 NEXT A
260 GOTO 260
270 END
```

Mode 10 has nine independent colors (each with a hue and luminance). It is a very special mode in that not all the color registers can be set using SETCOLOR commands. Instead, the registers are set by means of POKE statements (review Section 3.6). The memory locations which contain the color values are 704-712. Here is a list of what each register controls in mode 10:

REGISTER	USE
704	COLOR 0, BACKGROUND COLOR, BORDER COLOR
705	COLOR 1
706	COLOR 2
707	COLOR 3
708	COLOR 4 and 12
709	COLOR 5 and 13
710	COLOR 6 and 14
711	COLOR 7 and 15
712	COLOR 8, 9, 10, and 11

You poke these registers in memory by multiplying the hue (B) by 16 and adding the luminance (C) you would use in a normal SETCOLOR statement. Here is an example:

```
SETCOLOR 2,5,4 = POKE 710,84
```

because

```
5*16+4=84
```

Here is a program that uses mode 10 and "rotates" the colors:

```
5    PRINT "TYPE 8 NUMBERS SEPARATED BY
     COMMAS"
10   INPUT Q, W, E, R, T, Y, U, I
20   GRAPHICS 10
30   FOR A=1 TO 8
40     POKE 704+A,16*A+8
50   NEXT A
60   LET D=79:LET G=79:LET H=191:LET K=191
70   TRAP 130:REM -If an error occurs, GOTO
     line 130-
80   LET C=C+1:IF C=9 THEN LET C=1
90   COLOR C
100  PLOT A,S:DRAWTO D,F:DRAWTO G,H:DRAWTO
     J,K:DRAWTO A,S
110  LET A=A+Q:LET S=S+W:LET D=D-E:LET
     F=F+R:LET G=G-T:LET H=H-Y:LET J=J+U:LET
     K=K-I
120  GOTO 80
130  LET A=PEEK(705)
140  FOR B=705 TO 711
150    POKE B,PEEK(B+1)
160  NEXT B
170  POKE 712,A
180  FOR A=1 TO 20:NEXT A
190  GOTO 130
200  END
```

Mode 11 has sixteen hues and one luminance. It is ideal for making multi-colored pictures. SETCOLOR 4 controls the luminance for the 16 hues. Here is a program illustrating the multi-colored nature of mode 11:

```
10   GRAPHICS 11
20   LET C=0:S=1
30   FOR A=0 TO 191
40     C=C+S
50     COLOR C
60     IF C>16 OR C<1 THEN LET S=-S:GOTO 40
70     PLOT 0,A:DRAWTO 79,A
80   NEXT A
90   FOR A=0 TO 15
100    SETCOLOR 4,0,A
110    FOR D=1 TO 250
120    NEXT D
130  NEXT A
140  END
```

Modes 14-15

These modes can only be used on the ATARI XL computers. Both modes have text windows. Mode 14 is a two-color mode, while mode 15 is a four-color mode. Mode 15 is the four-color mode with the best resolution. It is probably the best mode for making highly detailed color pictures. You can think of mode 14 as mode 6 with 160×160 resolution instead of 160×80, and you can think of mode 15 as mode 7 with 160×160 resolution instead of 160×80.

ANSWER TO TEST YOUR UNDERSTANDING 1

1: Replace lines 30 and 60 with:

```
30 SETCOLOR 0,12,10:COLOR 1
60 SETCOLOR 2,1,6:COLOR 3
```

6.4 Text Commands

In this section we will discuss the commands used in conjunction with the text modes. These modes are modes 0, 1, 2, 12, and 13. Each of these modes can display only characters. The PLOT, DRAWTO, and COLOR statements can be used with these modes, but they have different meanings and uses than they do with graphics modes. Here are the statements we will explain in this section:

COLOR
DRAWTO
PLOT
POSITION
PRINT #6

Before we go into these statements in detail, let's consider Table 6-3.

The PLOT and DRAWTO statements are used much as they are with graphics. They can plot "points" and draw "lines," but instead of coloring pixels, they print characters. The COLOR statement is used to select which character will be used. Type this while in the direct mode:

Table 6–3. ATARI text modes.

Mode	*Resolution*	*# colors of text*	*SETCOLOR #*	*Use*	*Default value*
0	40 × 24	1 luminance	0	NO USE	Orange (2,8)
			1	Luminance of text	Light Green (12,10)
			2	Color (but no luminance) of text, Background color	Dark Blue (9,4)
			3	NO USE	Pink (4,6)
			4	Border color	Black (0,0)
1	20 × 20 w/window 20 × 24 w/o window	4	0	Color of color #32–95	Orange (2,8)
			1	Color of color #0–31 and #96–127, Luminance of text in text window	Light Green (12,10)
			2	Color of color #160–233, color of text window	Dark Blue (9,4)
			3	Color of color #128–159 and #224–255	Pink (4,6)
			4	Background color, border color	Black (0,0)
2	20 × 10 w/window 20 × 12 w/o window	4	0	Color of color #32–95	Orange (2,8)
			1	Color of color #0–31 and #96–127, Luminance of text in text window	Light Green (12,10)
			2	Color of color #160–233, color of text window	Dark Blue (9,4)
			3	Color of color # 128–159 and #224–255	Pink (4,6)
			4	Background color, border color	Black (0,0)
12	40 × 20 w/window 40 × 24 w/o window	4	0	Special	Orange (2,8)
			1	Luminance of text in text window, + special	Light Green (12,10)
			2	Color of text window, + special	Dark Blue (9,4)
			3	Special	Pink (4,6)
			4	Background color, border color	Black (0,0)
13	40 × 10 w/window 40 × 12 w/o window	4	0	Special	Orange (2,8)
			1	Luminance of text in text window, + special	Light Green (12,10)
			2	Color of text window, + special	Dark Blue (9,4)
			3	Special	Pink (4,6)
			4	Background color, border color	Black (0,0)

```
GRAPHICS 0
COLOR 65
PLOT 0,0:DRAWTO 20,20
```

The computer should have drawn a line made not of colored pixels, but of the character "A".

Each character you could have used to draw with has its own number to be used in a COLOR statement. The number used is called the ATASCII number of the character. The AT stands for ATARI and ASCII stands for American Standard Code for Information Interchange. Table 6-4 lists the ATASCII numbers and the characters they refer to. Figure 6-3 shows the graphics characters displayable by pressing CONTROL and a letter key.

Notice that in mode 0 the ATASCII number of an ordinary uppercase letter is between 65 and 90. The ATASCII number of the corresponding lowercase letter can be obtained by adding 32. Look at the characters in Table 6-4 with ATASCII numbers 0-31, 96, 123, and 125-127. With the exception of the ones numbered 27, 126, or 127, they are typed by pressing CTRL in conjunction with another key. They are called **control characters**.

If you add 128 to the ATASCII number of a character, you get the ATASCII number of the same character in **inverse video**; that is, with the colors of the foreground and background of the character interchanged. You obtain a character in inverse video by first pressing (but not holding down) the ATARI key and then pressing the key. The ATARI key is located at the left of the second SHIFT key on the 400 and 800, to the left of the BREAK key on the 1200XL and at the bottom right of the keyboard on the 600XL and 800XL. After typing in inverse video, press the ATARI key once again to return to normal. You cannot type in programs in inverse

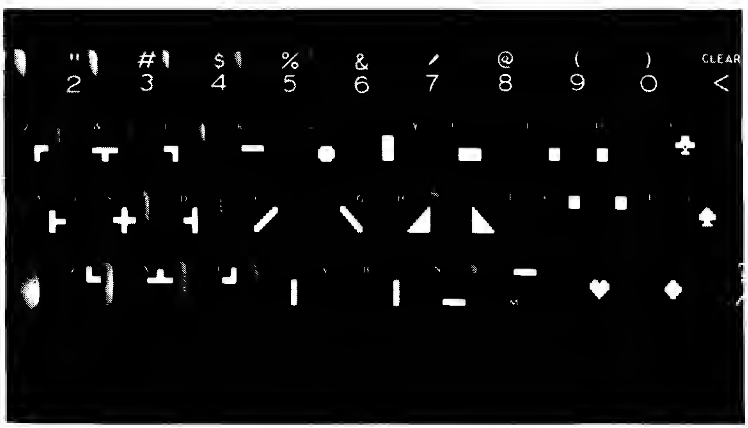

Figure 6-3. ATARI characters displayable using CONTROL.

Table 6-4. ATASCII codes.

Decimal Code	ATASCII Character	Keystrokes to Produce Character	Decimal Code	ATASCII Character	Keystrokes to Produce Character
0		CTRL-,	21		CTRL-U
1		CTRL-A	22		CTRL-V
2		CTRL-B	23		CTRL-W
3		CTRL-C	24		CTRL-X
4		CTRL-D	25		CTRL-Y
5		CTRL-E	26		CTRL-Z
6		CTRL-F	27	[1]	ESC\ESC
7		CTRL-G	28	[2]	ESC\CTRL--
8		CTRL-H	29	[3]	ESC\CTRL-=
9		CTRL-I	30	[4]	ESC\CTRL-+
10		CTRL-J	31	[5]	ESC\CTRL-*
11		CTRL-K	32		SPACE BAR
12		CTRL-L	33	!	SHIFT-1
13		CTRL-M	34	"	SHIFT-2
14		CTRL-N	35	#	SHIFT-3
15		CTRL-O	36	$	SHIFT-4
16		CTRL-P	37	%	SHIFT-5
17		CTRL-Q	38	&	SHIFT-6
18		CTRL-R	39	'	SHIFT-7
19		CTRL-S	40	(	SHIFT-9
20		CTRL-T	41	)	SHIFT-0

Table 6-4. ATASCII codes (continued).

Decimal Code	ATASCII Character	Keystrokes to Produce Character	Decimal Code	ATASCII Character	Keystrokes to Produce Character
42	*	*	63	?	SHIFT-/
43	+	+	64	@	SHIFT-8
44	,	,	65	A	A
45	-	-	66	B	B
46	.	.	67	C	C
47	/	/	68	D	D
48	0	0	69	E	E
49	1	1	70	F	F
50	2	2	71	G	G
51	3	3	72	H	H
52	4	4	73	I	I
53	5	5	74	J	J
54	6	6	75	K	K
55	7	7	76	L	L
56	8	8	77	M	M
57	9	9	78	N	N
58	:	SHIFT-;	79	O	O
59	;	;	80	P	P
60	<	<	81	Q	Q
61	=	=	82	R	R
62	>	>	83	S	S

Table 6-4. ATASCII codes (continued).

Decimal Code	ATASCII Character	Keystrokes to Produce Character	Decimal Code	ATASCII Character	Keystrokes to Produce Character
84	T	T	105	i	(LOWR) I
85	U	U	106	j	(LOWR) J
86	V	V	107	k	(LOWR) K
87	W	W	108	l	(LOWR) L
88	X	X	109	m	(LOWR) M
89	Y	Y	110	n	(LOWR) N
90	Z	Z	111	o	(LOWR) O
91	[	SHIFT- ,	112	p	(LOWR) P
92	\	SHIFT- +	113	q	(LOWR) Q
93	]	SHIFT- .	114	r	(LOWR) R
94	^	SHIFT- *	115	s	(LOWR) S
95	_	SHIFT--	116	t	(LOWR) T
96	♦	CTRL-.	117	u	(LOWR) U
97	a	(LOWR) A	118	v	(LOWR) V
98	b	(LOWR) B	119	w	(LOWR) W
99	c	(LOWR) C	120	x	(LOWR) X
100	d	(LOWR) D	121	y	(LOWR) Y
101	e	(LOWR) E	122	z	(LOWR) Z
102	f	(LOWR) F	123	♣	CTRL-;
103	g	(LOWR) G	124	|	SHIFT- =
104	h	(LOWR) H	125	↖[6]	ESC\CTRL-< or ESC\SHIFT-<

Table 6-4. ATASCII codes (continued).

Decimal Code	ATASCII Character	Keystrokes to Produce Character	Decimal Code	ATASCII Character	Keystrokes to Produce Character
126	[7]	ESC\BACK S	147		(Λ) CTRL-S
127	[8]	ESC\TAB	148		(Λ) CTRL-T
128		(Λ) CTRL-,	149		(Λ) CTRL-U
129		(Λ) CTRL-A	150		(Λ) CTRL-V
130		(Λ) CTRL-B	151		(Λ) CTRL-W
131		(Λ) CTRL-C	152		(Λ) CTRL-X
132		(Λ) CTRL-D	153		(Λ) CTRL-Y
133		(Λ) CTRL-E	154		(Λ) CTRL-Z
134		(Λ) CTRL-F	155	EOL[9]	(Λ) RETURN
135		(Λ) CTRL-G	156	[10]	ESC\SHIFT-BACK S
136		(Λ) CTRL-H	157	[11]	ESC\SHIFT->
137		(Λ) CTRL-I	158	[12]	ESC\CTRL-TAB
138		(Λ) CTRL-J	159	[13]	ESC\SHIFT-TAB
139		(Λ) CTRL-K	160		(Λ) SPACE BAR
140		(Λ) CTRL-L	161		(Λ) SHIFT-1
141		(Λ) CTRL-M	162		(Λ) SHIFT-2
142		(Λ) CTRL-N	163		(Λ) SHIFT-3
143		(Λ) CTRL-O	164		(Λ) SHIFT-4
144		(Λ) CTRL-P	165		(Λ) SHIFT-5
145		(Λ) CTRL-Q	166		(Λ) SHIFT-6
146		(Λ) CTRL-R	167		(Λ) SHIFT-7

Table 6-4. ATASCII codes (continued).

Decimal Code	ATASCII Character	Keystrokes to Produce Character	Decimal Code	ATASCII Character	Keystrokes to Produce Character
168	(	(Λ) SHIFT-9	189	=	(Λ) =
169	)	(Λ) SHIFT-0	190	>	(Λ) >
170	*	(Λ) *	191	?	(Λ) SHIFT-/
171	+	(Λ) +	192	@	(Λ) SHIFT-8
172	,	(Λ) ,	193	A	(Λ) A
173	-	(Λ) -	194	B	(Λ) B
174	.	(Λ) .	195	C	(Λ) C
175	/	(Λ) /	196	D	(Λ) D
176	0	(Λ) 0	197	E	(Λ) E
177	1	(Λ) 1	198	F	(Λ) F
178	2	(Λ) 2	199	G	(Λ) G
179	3	(Λ) 3	200	H	(Λ) H
180	4	(Λ) 4	201	I	(Λ) I
181	5	(Λ) 5	202	J	(Λ) J
182	6	(Λ) 6	203	K	(Λ) K
183	7	(Λ) 7	204	L	(Λ) L
184	8	(Λ) 8	205	M	(Λ) M
185	9	(Λ) 9	206	N	(Λ) N
186	:	(Λ) SHIFT-;	207	O	(Λ) O
187	;	(Λ) ;	208	P	(Λ) P
188	<	(Λ) <	209	Q	(Λ) Q

Table 6-4. ATASCII codes (continued).

Decimal Code	ATASCII Character	Keystrokes to Produce Character	Decimal Code	ATASCII Character	Keystrokes to Produce Character
210	R	(Λ) R	233	i	(Λ) (LOWR) I
211	S	(Λ) S	234	j	(Λ) (LOWR) J
212	T	(Λ) T	235	k	(Λ) (LOWR) K
213	U	(Λ) U	236	l	(Λ) (LOWR) L
214	V	(Λ) V	237	m	(Λ) (LOWR) M
215	W	(Λ) W	238	n	(Λ) (LOWR) N
216	X	(Λ) X	239	o	(Λ) (LOWR) O
217	Y	(Λ) Y	240	p	(Λ) (LOWR) P
218	Z	(Λ) Z	241	q	(Λ) (LOWR) Q
219	[	(Λ) SHIFT-,	242	r	(Λ) (LOWR) R
220	\	(Λ) SHIFT-+	243	s	(Λ) (LOWR) S
221	]	(Λ) SHIFT-.	244	t	(Λ) (LOWR) T
222	^	(Λ) SHIFT-*	245	u	(Λ) (LOWR) U
223	_	(Λ) SHIFT--	246	v	(Λ) (LOWR) V
224	♦	(Λ) CTRL-.	247	w	(Λ) (LOWR) W
225	a	(Λ) (LOWR) A	248	x	(Λ) (LOWR) X
226	b	(Λ) (LOWR) B	249	y	(Λ) (LOWR) Y
227	c	(Λ) (LOWR) C	250	z	(Λ) (LOWR) Z
228	d	(Λ) (LOWR) D	251	♠	(Λ) CTRL- ;
229	e	(Λ) (LOWR) E	252	|	(Λ) SHIFT- =
230	f	(Λ) (LOWR) F	253	↖ [14]	ESC\CTRL-2
231	g	(Λ) (LOWR) G	254	◀ [15]	(Λ) ESC\ CTRL-BACK S
232	h	(Λ) (LOWR) H	255	▶ [16]	(Λ) ESC\CTRL->

Table 6-4. ATASCII codes (continued).

Notes

[1]The character represents a control character. In most cases, this control character does nothing; CHR$(27) is generally a nondisplaying character. However, if the next character displayed is a control character with ATASCII codes 27, 28, 29, 30, 31, 125, 126, 127, 156, 157, 158, 159, 253, 254, or 255, the control process does not take place. Instead, the representative character itself appears.

[2]The character represents the control character which moves the cursor up one row. If the character displayed just before this was ATASCII code 27, the character displays; the cursor does not move.

[3]The character represents the control character which moves the cursor down one row. If the character displayed just before this was ATASCII code 27, the character displays; the cursor does not move.

[4]The character represents the control character which moves the cursor one column left. If the character displayed just before this was ATASCII code 27, the character displays; the cursor does not move.

[5]The character represents the control character which moves the cursor one column right. If the character displayed just before this was ATASCII code 27, the character displays; the cursor does not move.

[6]The character represents the control character which clears the screen and moves the cursor to the home position. If the character displayed just before this was ATASCII code 27, the character displays; the screen is not cleared.

[7]The character represents the control character which moves the cursor one column left and replaces the character there with a blank space. If the character displayed just before this was ATASCII code 27, the character displays; the cursor does not move.

[8]The character represents the control character which advances the cursor to the next tab stop. If the character displayed just before this was ATASCII code 27, the character displays; the cursor does not move.

[9]The ATASCII end-of-line character.

[10]The character represents the control character which deletes the line on which the cursor is located. If the character displayed just before this was ATASCII code 27, the character displays; the deletion does not occur.

[11]The character represents the control character which inserts a line above the one on which the cursor is located. If the character displayed just before this was ATASCII code 27, the character displays; the insertion does not occur.

[12]The character represents the control character which clears the tab stop (if any) at the current cursor position. If the character displayed just before this was ATASCII code 27, the character displays; no tab stop is affected.

[13]The character represents the control character which sets a tab stop at the current cursor position. If the character displayed just before this was ATASCII code 27, the character displays; no tab stop is set.

[14]The character represents the control character which beeps the built-in speaker; nothing is displayed. If the character displayed just before this was ATASCII code 27, the character displays; the speaker remains silent.

[15]The character represents the control character which deletes the character to the right of the cursor, shifting the remainder of the logical line one space to the left. If the character displayed just before this was ATASCII code 27, the character displays; no deletion occurs.

[16]The character represents the control character which inserts a blank space to the right of the cursor, shifting the remainder of the logical line one space to the right. If the character displayed just before this was ATASCII code 27, the character displays; no insertion occurs.

characters, but you can include inverse characters in a program, such as in REMs and PRINTs. When you press the ATARI key and then the space bar, a symbol resembling the cursor will appear. This symbol is not the cursor, just an **inverse space** character.

There are different characters which can be displayed in mode 0 and modes 1 and 2. In modes 1 and 2, any uppercase letter and many other characters can be in 4 colors. The characters available in these modes are listed in Table 6-4.

There are two commands that are related to ATASCII numbers. The ASC function gives the ATASCII number of a character. The format for this command is

```
PRINT ASC("X")
```

where X is any character. Here are some examples:

```
PRINT ASC("A")
PRINT ASC("p")
PRINT ASC("4")
PRINT ASC("♣")
```

The opposite of the ASC function is the CHR$ function. It returns the character of a specified ATASCII number. Here are some examples:

```
PRINT CHR$(65)
PRINT CHR$(112)
PRINT CHR$(52)
PRINT CHR$(16)
```

The POSITION command is used to put the cursor at a specific place on the screen. Remember, the location of the cursor is the place where the next piece of text will be written. Therefore, we could easily tell the computer to print HELLO in the middle of the screen. Here is the program to do that:

```
10 GRAPHICS 0
20 POSITION 17,11
30 PRINT "HELLO"
40 END
```

With graphics modes 1, 2, 12, and 13, you use PRINT only to print text in the text window. To print text on the screen but not in the text window, you would use the command PRINT #6 instead of PRINT. Here is the program to print HELLO in the middle of the screen in mode 1:

```
10 GRAPHICS 1
20 POSITION 7,9
30 PRINT #6;"HELLO"
40 END
```

Notice we put a semicolon after #6 in line 30 of the above program. (Either a semicolon or a comma is necessary with a PRINT #6 statement.) If we would have used a comma, the computer would have skipped 10 spaces before starting to print.

With modes 1 and 2, letters can be displayed only in uppercase. The letters may have any of the four colors stored in registers 0, 1, 2, and 3. When you type in a program, you must instruct the computer which of the four colors to use for each letter to be printed by PRINT #6. It would be much too complicated to use a COLOR statement for each letter. Instead, you type each letter in either uppercase or lowercase, regular or inverse video. Here is how to select the color for an uppercase letter printed by PRINT #6 in modes 1 and 2:

For the color in register:	Type the letter in:
0	Uppercase
1	Lowercase
2	Uppercase in inverse video
3	Lowercase in inverse video

The same sort of procedure is used for numerals, punctuation marks, and other symbols, but since there are no lowercase for these, only color registers 0 and 2 may be accessed with them (by typing just the symbol or the symbol in inverse video).

However, there is a way to display these symbols in four colors. Let's explain how to get the other two colors. Suppose you wanted to print an exclamation mark in the other two colors. If you look near the beginning of Table 6-4 you will find the exclamation mark in the second line. To the left of it, you will see the control symbol you should use to obtain the character. To the right of it, you will see that color register 1 is used to display it. If you key in the control symbol in a PRINT #6 statement with mode 1 or 2, an exclamation mark with the hue and luminance in color register 1 will be printed. The table gives you the keystrokes necessary, namely, CTRL-A (CONTROL-A on the XL computers). If you key the control character in inverse video, the exclamation mark will be displayed with the hue and luminance of color register 3. Of course, the color in any color register may be set using SETCOLOR.

6.5 Text Modes

The text modes are modes 0, 1, 2, 12, and 13. Modes 12 and 13 are available for the ATARI XL computers only. Table 6-3 on page 150 shows the resolution, number of colors, and the use of the five SETCOLOR numbers for each of these modes.

Mode 0

You are already familiar with this mode since it is the default mode. However, it can be used for more than printing ordinary text. In fact, mode 0 will be used in Chapter 8 for our games. For now, let's illustrate the possibilities of mode 0 by moving an arrow across the screen:

```
10  GRAPHICS 0
20  POKE 752,1:REM -Turns cursor off-
30  FOR X=1 TO 38
40    POSITION X,11
50    PRINT "->":REM -Print arrow-
60    POSITION X-1,11
70    PRINT " ":REM -Erase old tail-
80    FOR D=1 TO 45:NEXT D
90  NEXT X
100 GRAPHICS 0
110 POSITION 17,11
120 PRINT "THE END"
130 END
```

In mode 0, the color of the background is controlled by SETCOLOR 2. The text has the same hue as the background, but its luminance is controlled by SETCOLOR 1.

Mode 1

This mode differs in many ways from mode 0:

1. The text can be multi-colored and the colors are independent of the background color.
2. The characters are twice as wide.
3. There is a text window.

The following program illustrates how color can be used in mode 1 to create the illusion of motion:

```
10 GRAPHICS 1+16:A=1
20 PRINT #6;"J**J**J**J**J**J**J"
30 PRINT #6;"*                 *"
40 PRINT #6;"*                 *"
50 PRINT #6;"J                 J"
60 PRINT #6;"* happy birthday  *"
70 PRINT #6;"*                 *"
80 PRINT #6;"J                 J"
90 PRINT #6;"*                 *"
100 PRINT #6;"*                 *"
110 PRINT #6;"J**J**J**J**J**J**J"
160 SETCOLOR 0,2,8
170 SETCOLOR 1,0,0
180 SETCOLOR 2,9,4
190 FOR D=1 TO 75:NEXT D
200 SETCOLOR 1,2,8
210 SETCOLOR 2,0,0
220 SETCOLOR 0,9,4
230 FOR D=1 TO 75:NEXT D
240 SETCOLOR 2,2,8
250 SETCOLOR 0,0,0
260 SETCOLOR 1,9,4
270 FOR D=1 TO 75:NEXT D
280 GOTO 160
290 END
```

Mode 2

The only difference between modes 1 and 2 is that the characters in mode 2 are twice as tall as in mode 1. To see the the effect this can have, change line 10 of the preceding program so it becomes

```
10 GRAPHICS 2+16
```

and run the new program.

Modes 12 and 13

These are four-color modes with text windows. The characters in mode 12 have the same size as those in mode 0, and the characters in mode 13 are twice as high. In these modes, it is possible to display up to four colors in one character. This makes these two modes especially suitable for animation. Since advanced programming techniques are required for animation, we will not pursue

the topic. For more information about this subject and others, a good reference book is DE RE ATARI by Chen et al.

6.6 More Commands

In this section we will discuss the XIO and LOCATE commands, which are useful commands in conjunction with the display modes.

The XIO Command

Although the XIO command has many uses, we will discuss only its use to fill in a portion of the screen. In graphics modes, it fills in with pixels; in text modes, it fills in with characters. The XIO command for filling parts of the screen is always the same:

```
XIO 18,#6,0,0,"S:"
```

However, the command must be used in conjunction with other commands. After some PLOT and DRAWTO statements have been given to determine the right boundary of the region to be filled, the format is

```
POKE 765,N
POSITION X,Y
XIO 18,#6,0,0,"S:"
```

Here N is the color number of the "filling." In a graphics mode, it determines the color of the region to be filled. In a text mode, it is

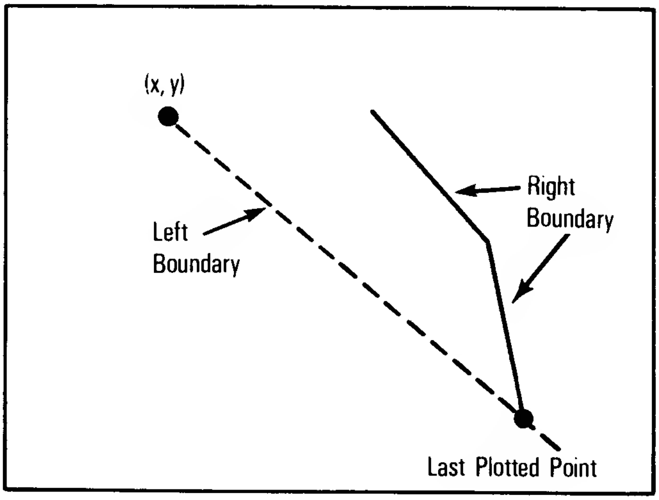

Figure 6-4.

the ATASCII number of the character used to fill in the region. X and Y are the coordinates of a point.

Here is what happens: Imagine a straight line that joins the last plotted point to the point X,Y. That line is the left boundary of the region to be filled in. Starting at the last plotted point and moving along the left boundary toward X,Y, the computer fills in along horizontal lines from the left boundary to the right boundary (see Figure 6-4). If any of these horizontal lines fills all the way to the right edge of the screen without hitting the right boundary, it will wrap around to the left side of the screen and continue until it hits the right boundary of the region or returns to its starting point.

For our first example, let's fill in a triangular region in mode 8+16 (mode 8 without a text window):

```
10  GRAPHICS 8+16
20  SETCOLOR 2,0,0:REM -Turn screen black for
    better visibility-
30  COLOR 1
40  PLOT 310,10
50  DRAWTO 310,181
60  POKE 765,1
70  POSITION 10,10
80  XIO 18,#6,0,0,"S:"
90  GOTO 90
100 END
```

Notice we used the same color number for filling and plotting (see lines 30 and 60). If we insert the line

```
55 PLOT 10,181
```

the last plotted point before the XIO command will be on the lower left of the screen, and the filled in region will be a rectangle. Add line 55 to the program and run it again to compare the two regions.

Now let's fill in some regions which are more complicated. The following program uses two XIO statements to fill in a region that resembles a house:

```
10  GRAPHICS 8+16
20  SETCOLOR 2,0,0
30  COLOR 1
40  PLOT 310,78:DRAWTO 310,181
50  PLOT 10,181
60  POSITION 10,78
70  POKE 765,1
80  XIO 18,#6,0,0,"S:"
90  PLOT 310,78:DRAWTO 160,10
100 PLOT 10,78
```

```
110 POSITION 160,10
120 XIO 18,#6,0,0,"S:"
130 GOTO 130
140 END
```

We can add two windows and a door to the house by using five XIO statements:

```
10  GRAPHICS 8+16
20  SETCOLOR 2,0,0
30  COLOR 1
40  PLOT 135,180:DRAWTO 135,150:REM -Draw
    door-
50  PLOT 80,100:DRAWTO 80,120:REM -Draw first
    window-
60  PLOT 200,100:DRAWTO 200,120:REM -Draw
    second window-
70  PLOT 310,78:DRAWTO 310,181:REM -Draw
    right wall of house-
80  PLOT 10,181
90  POSITION 10,78
100 POKE 765,1
110 XIO 18,#6,0,0,"S:":REM -Fill in most of
    house-
120 PLOT 310,78:DRAWTO 160,10:REM -Draw roof-
130 PLOT 10,78
140 POSITION 160,10
150 XIO 18,#6,0,0,"S:":REM -Fill in roof-
160 PLOT 165,180
170 POSITION 165,149
180 XIO 18,#6,0,0,"S:":REM -Fill in right of
    door-
190 PLOT 100,121
200 POSITION 100,99
210 XIO 18,#6,0,0,"S:":REM -Fill in part
    between windows-
220 PLOT 220,121
230 POSITION 220,99
240 XIO 18,#6,0,0,"S:":REM -Fill in right of
    2nd window-
250 GOTO 250
260 END
```

When filling in complicated figures (like the house with windows and a door), it is necessary to use several XIO commands. To simplify writing the program, you can write a program with the first

XIO command, and run that program to see how things look. Then add another XIO command to fill in another part of the screen. By repeating this procedure, you will eventually have a program which fills exactly what you want.

We have provided two programs to illustrate how flexible the XIO command is. The programs allow you to draw a highway using either a joystick or paddle. If you are not familiar with joysticks or paddles yet, come back to these two programs when you finish Chapter 7.

Joystick version:

```
10   GRAPHICS 8+16
20   SETCOLOR 2,0,0
30   COLOR 1
40   LET X=135
50   FOR Y=0 TO 191
60     S=STICK(0)
70     IF S=15 THEN 60
80     IF S=11 THEN X=X-1
90     IF S=7 THEN X=X+1
100    PLOT X,Y
110    PLOT X+50,Y:DRAWTO 319,Y
120    IF Y/2=INT(Y/2) THEN PLOT
       X+24,Y:DRAWTO X+26,Y
130 NEXT Y
140 PLOT 0,0
150 POSITION 0,191
160 POKE 765,1
170 XIO 18,#6,0,0,"S:"
180 GOTO 180
190 END
```

Paddle version:

```
10   GRAPHICS 8+16
20   SETCOLOR 2,0,0
30   COLOR 1
40   FOR Y=0 TO 191
50     FOR D=1 TO 50
60     NEXT D
70     X=228-PADDLE(0)
80     PLOT X,Y
90     PLOT X+50,Y:DRAWTO 319,Y
100    IF Y/2=INT(Y/2) THEN PLOT
       X+24,Y:DRAWTO X+26,Y
```

```
110 NEXT Y
120 PLOT 0,0
130 POSITION 0,191
140 POKE 765,1
150 XIO 18,#6,0,0,"S:"
160 GOTO 160
170 END
```

With a little practice you should be able to use either of these programs to create the illusion of a highway on a hilly terrain.

The LOCATE Command

Using the PRINT or PLOT command, you can put a character or a pixel on the screen. The LOCATE command allows you to "examine" a location on the screen to discover what character or color is in that location. It assigns a number to a variable specified by you in the command. The form of the statement is

```
LOCATE X,Y,C
```

where X,Y are the coordinates of the location to be examined and C is the variable specified by you. The value assigned to C is the color number of the pixel or character. In a four-color graphics mode, C will be 0, 1, 2, or 3. In a two-color graphics mode, C will be 0 or 1. In a text mode, C will be the ATASCII number of the character (see Table 6-4 on page 152).

As a simple example, we will print the letter J (whose ATASCII number is 74) at 10,12 in GRAPHICS 1 and then use the LOCATE statement to have the computer find the ATASCII number of the character at 10,12 and use it to print out the character. Of course, we know the result will be 74, but the program is for illustration only.

```
10 GRAPHICS 1
20 COLOR 74
30 PLOT 10,12
40 LOCATE 10,12,C
50 PRINT "THE CHARACTER AT 10,12 IS ";CHR$(C)
60 END
```

For a more complicated example, consider the following program:

```
10  GRAPHICS 0
20  POKE 752,1
30  PRINT "                A" (SKIP 16 SPACES
    BEFORE A.)
40  PRINT "               ACA" (SKIP 15
    SPACES BEFORE ACA...ETC.)
50  PRINT "              ACCCA"
60  PRINT "             ACCACCA"
70  PRINT "            ACCACACCA"
80  PRINT "           ACCACACACCA"
90  PRINT "          ACCACACACACCA"
100 PRINT "         ACCAACACACAACCA"
110 PRINT "         ACCCCCCCCCCCCCA"
120 PRINT "         ACCCCCCCCCCCCCA"
130 PRINT "         ACCAACACACAACCA"
140 PRINT "          ACCACACACACCA"
150 PRINT "           ACCACACACCA"
160 PRINT "            ACCACACCA"
170 PRINT "             ACCACCA"
180 PRINT "              ACCCA"
190 PRINT "               ACA"
200 PRINT "                A"
210 FOR Y=0 TO 17
220   FOR X=11 TO 25
230     LOCATE X,Y,C
240     IF CHR$(C)="A" THEN POSITION
        X,Y:PRINT CHR$(16)
250     IF CHR$(C)="B" THEN POSITION
        X,Y:PRINT CHR$(20)
260     IF CHR$(C)="C" THEN POSITION
        X,Y:PRINT CHR$(160)
270   NEXT X
280 NEXT Y
290 POKE 752,0
300 END
```

In the first part of the program we create a picure, using the letters A, B, and C. Then, with the help of the LOCATE and IF . . . THEN commands, we change the letters A, B, and C to other characters to obtain a design.

The LOCATE command is often used in simple games to determine whether your "ship" hit a wall or some other object.

7

Using Sound and Game Controllers

7.1 Using Sound

ATARI computers have one of the best sound chips available for home computers. You can synthesize many sounds such as waves, explosions, and "pure" tones. You can produce 4 different sounds at the same time. Each sound is called a **voice**. For instance, you could have 2 voices playing a song and the other two producing breaking waves in the background.

To produce sounds on the ATARI, you use the SOUND statement. It has the following form:

```
SOUND <VOICE>, <NOTE>, <TONE>, <VOLUME>
```

Example: `SOUND 0, 121, 10, 10`

Let us explain VOICE, NOTE, TONE, and VOLUME. As we already stated, you may produce up to four individual sounds, or voices, simultaneously. These are numbered 0, 1, 2, and 3. If you wish to produce a single sound, you may use any of these numbers for VOICE. If you wish to make several sounds, you give a separate SOUND statement for each, and use different numbers for VOICE. Once a voice is turned on, it will stay on until you turn it off by another SOUND statement with the same voice number or by an END statement. For example, SOUND 1,0,0,0 turns off voice 1.

NOTE can range from 0 to 255. It determines the pitch of the sound produced. Table 7-1 shows the musical notes certain NOTE numbers produce:

Table 7-1. The musical notes produced by NOTE numbers.

NOTE number	*Musical note*
29	C
31	B

33	A# or B♭
35	A
37	G# or A♭
40	G
42	F# or G♭
45	F
47	E
50	D# or E♭
53	D
57	C# or D♭
60	C
64	B
68	A# or B♭
72	A
76	G# or A♭
81	G
85	F# or G♭
91	F
96	E
102	D# or E♭
108	D
114	C# or D♭
121	Middle C
128	B
136	A# or B♭
144	A
153	G# or A♭
162	G
173	F# or B♭
182	F
193	E
204	D# or E♭
217	D
230	C# or D♭
243	C

TONE can be any even number from 0 to 14. It represents the amount of distortion of the sound. The numbers 10 and 14 produce pure tones.

VOLUME can also be any even number from 0 to 14. It determines the loudness of the sound. 14 is loud and 0 is off. 8 is an average number.

Now type in and run the following program:

```
10 FOR NOTE=0 TO 254
20   SOUND 0,NOTE,10,10
```

```
30 NEXT NOTE
40 END
```

The above program uses only one voice and changes only the pitch. It does not change the distortion or volume. Here is a program that changes only the volume:

```
10 FOR VOLUME=14 TO 0 STEP -2
20   SOUND 0,121,10,VOLUME
30   FOR DELAY1=1 TO 350
40   NEXT DELAY1
50   SOUND 0,0,0,0:REM -Turns off sound
60   FOR DELAY2=1 TO 150
70   NEXT DELAY2
80 NEXT VOLUME
90 END
```

This program changes the volume of a certain note. Notice the delay loops. If they were omitted, the sound would change so fast you couldn't hear the individual sounds very well. Here is a program that changes only the distortion:

```
10 FOR DISTORTION=0 TO 14 STEP 2
20   SOUND 0,10,DISTORTION,10
30   FOR DELAY1=1 TO 350
40   NEXT DELAY1
50   SOUND 0,0,0,0:REM -Turns off the sound-
60   FOR DELAY2=1 TO 150
70   NEXT DELAY2
80 NEXT DISTORTION
90 END
```

The preceding program does not produce very pleasant sounds, but it does illustrate what different distortions can do.

TEST YOUR UNDERSTANDING 1 (answers on page 173)

In the statement

```
SOUND A,B,C,D
```

which variable stands for volume, which for pitch, which for voice, and which for distortion.

Here are some programs that produce some interesting sound effects:

BREAKING WAVES

```
10  FOR NOTE1=5 TO 20
20    SOUND 0,NOTE1,0,14
30    FOR DELAY1=1 TO 100
40    NEXT DELAY1
50  NEXT NOTE1
60  FOR NOTE2=20 TO 5 STEP -1
70    SOUND 0,NOTE2,0,10
80    FOR DELAY2=1 TO 75
90    NEXT DELAY2
100 NEXT NOTE2
110 END
```

BOUNCING BALL

```
10  TIME=500
20  SOUND 0,15,2,10
30  FOR DELAY1=1 TO 10
40  NEXT DELAY1
50  SOUND 0,0,0,0:REM -Turn off sound-
60  FOR DELAY2=1 TO TIME
70  NEXT DELAY2
80  TIME=TIME/1.5
90  IF TIME<1 THEN END
100 GOTO 20
110 END
```

DROPPING BOMBS

```
10 FOR PITCH=50 TO 255
20   SOUND 0,PITCH,10,4
30 NEXT PITCH
40 FOR VOLUME=14 TO 0 STEP -0.1
50   SOUND 0,255,0,VOLUME
60   SOUND 1,10,0,VOLUME
70   SOUND 2,150,2,VOLUME
80 NEXT VOLUME
90 END
```

EMERGENCY BROADCASTING SYSTEM

```
10 SOUND 0,28,10,10
20 SOUND 1,32,10,10
30 GOTO 30
40 END
```

PINBALL MACHINE

```
10  FOR NOTE1=0 TO 15
20    SOUND 0,NOTE,2,10
30    FOR DELAY=1 TO 10
40    NEXT DELAY
50  NEXT NOTE1
60  FOR NOTE2=14 TO 0 STEP -1
70    SOUND 0,NOTE2,2,10
80    FOR DELAY=1 TO 10
90    NEXT DELAY
100 NEXT NOTE2
110 END
```

With a little practice you will be able to produce sound to go along with your own programs.

ANSWER TO TEST YOUR UNDERSTANDING 1

1: D stands for volume
B stands for pitch
A stands for voice
C stands for distortion

7.2 Using Joysticks

A joystick is used for most arcade games and can be used with ATARI computers, too. The standard ATARI joystick is shown in Figure 7-1.

Frequently, a joystick is used in games to move something around on the screen. The direction you push the lever on the joystick determines the direction of motion on the screen. If you have an ATARI joystick, it should be held in such a way that the red button is in the upper left corner.

In this section, we will explain how to use joysticks. The ATARI 400 and 800 have four jacks (numbered 1-4) for connecting joysticks (see Figures 7-2 and 7-4, respectively). The ATARI 600XL and 800XL have two jacks, numbered 1 and 2, on the right side of the computer (see Figures 7-3 and 7-5, respectively). The ATARI 1200XL has two jacks, numbered 1 and 2, on the left side of the computer (see Figure 7-6).

For this section we suggest you plug your joystick in jack 1 of your computer. Then, type in this program:

Figure 7-1. A standard ATARI joystick.

```
10 PRINT STICK(0)
20 GOTO 10
```

Notice the 0 in STICK(0). It refers to the joystick in jack 1. Whenever the computer refers to a joystick, it always uses the number that is one less than the number of the jack into which the joystick is plugged. Now type RUN and start moving the joystick around. Whenever you push the lever on the joystick in any direction, the computer will print a number corresponding to the direction. When you are not pushing the lever on the joystick in any direction or there is no joystick plugged in, the computer will print out the number 15. There are eight positions in which you can push the lever: left, right, up, down, and the four diagonal directions. When the computer encounters the statement

```
PRINT STICK(0)
```

in a program, it will return a value depending upon the direction in which you are pushing the lever on the joystick in jack 1. Figure 7-7 shows the top view of an ATARI joystick with the numbers an ATARI computer associates with the various directions the lever can be pushed. We will call these the **direction numbers**.

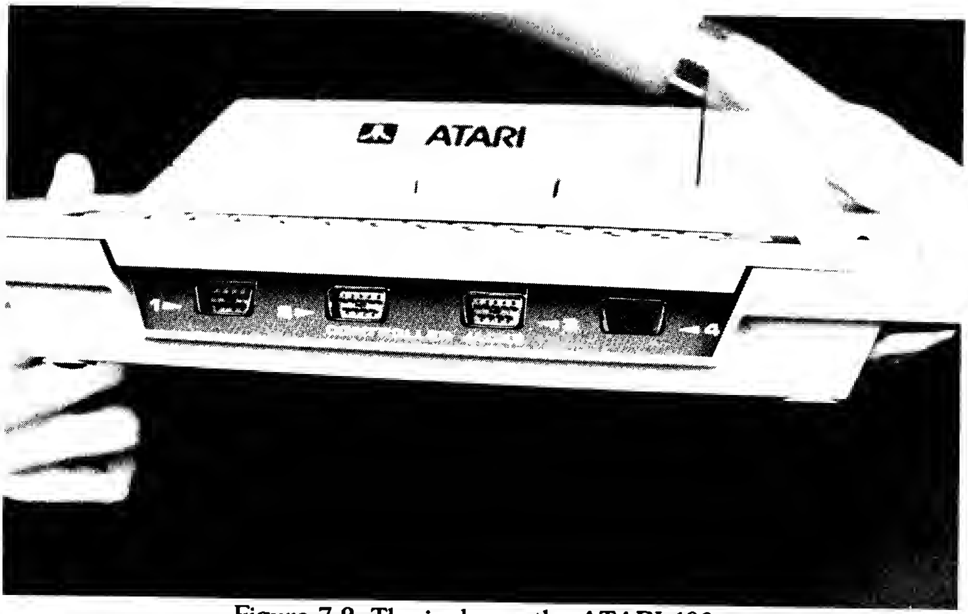

Figure 7-2. The jacks on the ATARI 400.

If a joystick is in jack 1, STICK(0) gives the direction number of that joystick. STICK(1) gives the direction number of the joystick in jack 2, and so on. Naturally, it is possible to use more than 1 joystick in a program. Shortly, we will present a program that

Figure 7-3. The jacks on the ATARI 600XL.

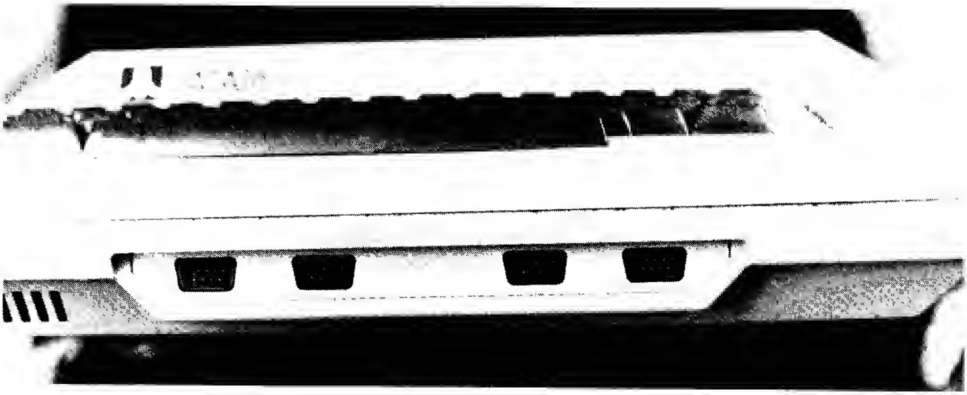

Figure 7-4. The jacks on the ATARI 800.

Figure 7-5. The jacks on the ATARI 800XL.

illustrates how the joystick can be used. But, first, let us point out that you can assign a variable to STICK(0) as follows:

```
10 LET D=STICK(0)
```

Then you could use D throughout a program for the direction number. Of course, you could use any other variable in place of D and make the assignment on any other line of the program, if you wished.

Here is our program. It will allow you to move a plus sign around the screen. Of course, you don't have to type the REM statements in.

```
10  GRAPHICS 0:REM -Clear the screen-
20  POKE 752,1: REM -Turn off cursor-
30  LET X=20:REM -Initial horizontal position
    of plus sign-
```

```
40  LET Y=12:REM -Initial vertical position
    of plus sign-
50  LET D=STICK(0):REM -Direction of joystick
    #1
60  IF D=11 THEN LET X=X-1:REM -move left-
70  IF D=7 THEN LET X=X+1:REM -move right-
80  IF D=14 THEN LET Y=Y-1:REM -move up-
90  IF D=13 THEN LET Y=Y+1:REM -move down-
100 IF D=10 THEN LET X=X-1:LET Y=Y-1:REM -
    Move up and left-
110 IF D=6 THEN LET X=X+1:LET Y=Y-1:REM -Move
    up and right-
120 IF D=9 THEN LET X=X-1:LET Y=Y+1:REM -Move
    down and left-
130 IF D=5 THEN LET X=X+1:LET Y=Y+1:REM -Move
    down and right-
140 IF X<0 THEN LET X=0:REM -Stop at left
    margin-
150 IF X>38 THEN LET X=38:REM -Stop at right
    margin-
160 IF Y<0 THEN LET Y=0:REM -Stop at top
    margin-
170 IF Y>21 THEN LET Y=21:REM -Stop at bottom
    margin-
180 IF X<>OX OR Y<>OY THEN POSITION
    OX,OY:PRINT " ":REM -Erase last position
190 POSITION X,Y:PRINT "+":REM -Print new
    plus sign-
```

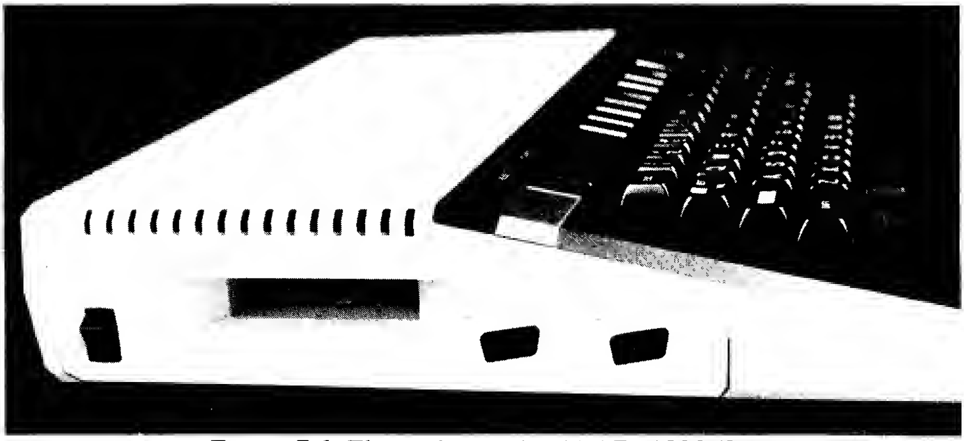

Figure 7-6. The jacks on the ATARI 1200XL.

```
200 LET OX=X:LET OY=Y:REM -Remember current
    position-
210 GOTO 50:REM -Loop-
```

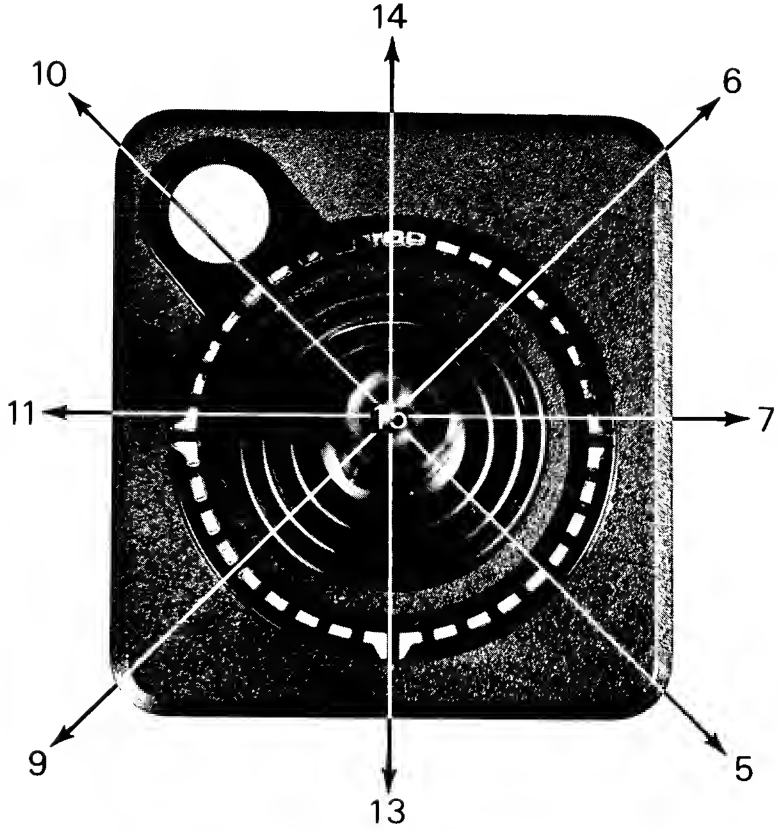

Figure 7-7. The top view of an ATARI joystick.

Type in the preceding program and run it. Plug a joystick in jack 1 and move your plus sign around. Let us explain briefly what the individual parts of the program do:

- Lines 30-40 set the the initial position of the plus sign to the center of the screen.
- Line 50 detects the direction in which the lever on the joystick is pushed.
- Lines 60-130 determine the coordinates of the position to which the plus sign is to be moved (according to the joystick).
- Lines 140-170 ensure that the plus sign doesn't go off the screen.

- Line 180 erases the plus sign from the old position (if the new position is different from the old position).
- Line 190 prints a plus sign at the new position.
- Line 200 "remembers" the current position of the plus sign.
- Line 210 takes us back to line 50 and the process begins again.

Make sure you understand how this program works. You may have to read this section a few times, but you should be able to grasp it.

At this point, you have probably noticed a small red button at the top left corner of your joystick. This is called the **trigger** or **fire button**. In games, it is most commonly used much like the trigger of a gun. When you press the trigger, you will shoot bullets, drop bombs, launch rockets, etc.

You detect whether the button is pressed much as you detect whether the lever on the joystick is pushed in a certain direction. Try typing this while holding down the button of your joystick:

```
PRINT STRIG(0)
```

The computer should have printed "0" on the screen. When you type PRINT STRIG(0), the computer will either return a 1 or 0. 1 means the button is not being pressed and 0 means it is. We can detect whether the trigger of a joystick plugged in any jack is being pressed by typing

```
PRINT STRIG(X)
```

where X is one less than the jack number. We will now add some line numbers to the previous program. Type in each of these lines:

```
205 IF STRIG(0)=0 THEN GOSUB 220: REM -If
    trigger is pressed, jump to the
    subroutine at line 220-
220 POSITION 0,22:PRINT "X=";X;" "
230 POSITION 5,22:PRINT "Y=";Y;" "
240 FOR A=0 TO 30
250   SOUND 0,A,10,10
260 NEXT A
270 SOUND 0,0,0,0
280 RETURN
290 END
```

Now run the program with the changes. Try pressing the joystick trigger. You will see the X (horizontal) and Y (vertical) coordinates of your plus sign. You will also hear a sound if the volume on your TV is turned up. Make sure you understand this program. If you don't, reread this section.

TEST YOUR UNDERSTANDING 1 (answers on page 180)

a. Write a program that prints the direction the lever on a joystick in jack 2 is being pressed.
b. Write another program that detects whether the trigger on a joystick in jack 2 is being pressed.

ANSWERS TO TEST YOUR UNDERSTANDING 1

1: a.

```
10  LET D=STICK(1)
20  PRINT "THE DIRECTION IS ";
30  IF D=14 THEN PRINT "UP"
40  IF D=6 THEN PRINT "UPPER RIGHT"
50  IF D=7 THEN PRINT "RIGHT"
60  IF D=5 THEN PRINT "LOWER RIGHT"
70  IF D=13 THEN PRINT "DOWN"
80  IF D=9 THEN PRINT "LOWER LEFT"
90  IF D=11 THEN PRINT "LEFT"
100 IF D=10 THEN PRINT "UPPER LEFT"
110 GOTO 10
120 END
```

b.

```
10 LET S=STRIG(1)
20 IF S=0 THEN PRINT "THE TRIGGER IS
   BEING PRESSED."
30 IF S=1 THEN PRINT "THE TRIGGER IS
   NOT BEING PRESSED."
40 GOTO 10
50 END
```

7.3 Using Paddles

In this section, we will explain how to use an ATARI paddle, another kind of game controller. If you have not read Section 7.2, please do so now. This section will be much easier to comprehend if you understand joysticks. A set of standard ATARI paddles is shown in Figure 7-8.

Paddles plug into the same ports as joysticks. We will assume you have a paddle plugged into jack 1. Notice there are 2 paddles for 1

Figure 7-8. A set of standard ATARI paddles.

connector. The one on the left is paddle 0. The other one is paddle 1. We suggest you put a piece of tape on paddle 0 so you can find it easily. The paddles in jack 2 are paddles 2 and 3, and so on.

Paddles in arcades are usually used for moving an object along a specific path. For instance, in the game called BREAKOUT, you use a paddle to move a small barrier along the bottom of the screen.

Try typing in and running this program:

```
10 PRINT PADDLE(0),
20 PRINT PADDLE(1)
30 GOTO 10
40 END
```

If you turn the knobs on the paddles all the way to the left, 228 will be displayed. If you turn the knobs on the paddles all the way to the right, 0 will be displayed.

You detect whether the trigger on the paddle is pressed much as you detect whether the trigger on the joystick is pressed, but instead of using PRINT STRIG(0), you use PRINT PTRIG(0). The

number 1 means the trigger is not being pressed and 0 means it is. Type in the following program and run it:

```
10 IF PTRIG(0)=0 THEN PRINT "TRIGGER PRESSED"
20 GOTO 10
30 END
```

When you press the trigger on paddle 0, the computer should print TRIGGER PRESSED on the screen until you release the trigger. If you are not pressing the trigger, the computer will not display anything. Here is a program that illustrates how a paddle can be used:

```
10 LET R=228/39.99
20 LET P=PADDLE(0)
30 LET X=INT(P/R)
40 POSITION X,23
50 PRINT "+"
60 GOTO 20
70 END
```

Since PADDLE(0) can be as large as 228, the number P/R will be between 0 and 39.99. Therefore, X will be an integer between 0 and 39. Lines 30 and 40 will print a plus sign on the bottom line of the screen according to how far the dial on the paddle is turned. Each time something is to be printed on the bottom line of the screen, the computer automatically "scrolls" the screen upward to make room for a new line. The top line is pushed off the screen, the second line is pushed to the top, and so forth. To see this effect, plug a paddle into jack 1, run the program, and turn the knob on paddle 0.

8

Games

8.1 How to Make Your Own Games

In the last few chapters we learned about graphics, sound, joysticks, paddles, and so forth, but we haven't really applied them to making a good program that might impress someone. In this chapter we will explain how to make a game, and even provide some games you can type in and play.

If you have ever visited a computer store or read a computer magazine, you probably know how popular and well advertised computer games are. Most games are also costly, to say the least. The average price of a game is about $29.95. If you are good at programming, you could make your own games and, perhaps, sell them.

When making a game, there is one thing you must never forget: ORGANIZATION. Without some kind of organization, it is very difficult, if not impossible, to make a game. Before you do anything, you must think of a game you want to make. You could modify an existing game, or create a totally new one. Once you have your game idea, make a list of everything that moves in the program. This can include non-living objects, such as trains, rocks, and so forth.

For example, imagine a game in which you are a frog. You chase flies, and snakes chase you. The list of moving objects would be:

1. You (the frog)
2. Snakes
3. Flies

After you make this list, imagine the game in play. Think about what the computer must do to make the game work correctly. Think about the order of the movements of the objects in the table above. Think about what each one does while it moves. Make a general outline including these facts. Here is an example of such a general outline:

I. Move the frog
  A. Check if it is on a log
    1. No, end game
II. Move snakes
  A. Check if any snake collides with frog
    1. Yes, end game
III. Move flies
  A. Check if frog collides with a fly
    1. Yes, add 10 to nutrition
    2. No, subtract 1 from nutrition
  B. Check if nutrition is below 0
    1. Yes, end game
IV. GOTO step 1

This list will help you to form the general structure of your program.

The first list was intended to help you make the second list. Now that you have the second list down, you should make a variable list. This is a list which tells you what variables are going to be used in your program. You may want to add to this list while writing a program. This is perfectly fine.

VARIABLE	USE
A	General purpose variable
D	Delay loops
N	Nutrition
FX	X-coordinate of frog
FY	Y-coordinate of frog
FLX(30)	X-coordinate of flies
FLY(30)	Y-coordinate of flies
SX(10)	X-coordinate of snakes
SY(10)	Y-coordinate of snakes

If your program will contain many subroutines, write down each subroutine's line number and use in a table like this:

SUBROUTINE LINE	USE
1000	Make a jumping sound
2000	Make a crunching sound when you get a fly
3000	"Kill" frog
4000	Set up screen color, draw logs, etc.

This will help you keep track of where to go for a specific subroutine.

Now that you have completed this table, you have two choices: either make your program right now or make a very specific outline (or flowchart). We discussed flowcharts earlier in the book. It will help beginners, but if you think you are good enough at programming to skip this step, go ahead. The choice if yours, but make sure you pick the option which will help you most.

After you complete all these tables and lists, you are ready to sit down at your computer and start programming. A good idea is to save the program on disk or cassette every page or so of writing. This is to prepare for sudden "black outs" of the computer. This happens once in a while. If you follow all the steps provided, your game will probably work with only minor debugging. Here is a summary of the steps involved in making a game:

1. Think of a topic.
2. Make a list of everything which moves.
3. Make a general outline of your program.
4. Make a variable list.
5. Make a table of the subroutines in your program.
6. Make a flowchart of your program (optional).
7. Write the program on the computer, remembering to save the program occasionally.
8. Debug the program if necessary.

> **NOTE:** NEVER use your computer during a storm unless you have a power surge protector that protects your computer from surges and dips in electricity, often caused by lightning. Make sure there isn't too much static electricity in the room. Even a small spark can erase and destroy your computer's memory forever!

8.2 Tic Tac Toe

In Section 8.1, we discussed how to make a game. In this and the next section, we will actually provide games you can type in and play. We will provide all the tables and lists we used for making the games.

In this section we provide a version of the popular tic tac toe game. The program is written so that two people play the game. The computer makes sure you don't cheat (putting a mark in an already occupied square) and it tells you when someone has won

the game. You specify which box you want to put your mark in by giving a number from 1-9. There is a small table next to the tic tac toe board that shows what each number means.

Here are the tables and lists involved in the program.

General outline:

I. Draw tic tac toe board
II. Ask players' names
  A. Ask first player's name
  B. Ask second player's name
III. Determine which player moves first
IV. Ask for move of player up next
V. Check for validity of move
VI. Plot X or O in square
VII. Check if person won
  Yes—ask if another game should be played
    Yes—GOTO step I
    No—end game
  No—GOTO step IV

Variable list:

variable	*use*
A	General purpose
B	General purpose
D	Delay
F	Tells whose turn it is
M	Number (coordinate) of person's move
A$	General purpose string
P1$	Name of player 1
P2$	Name of player 2
G(3,3)	Stores characters which are in the tic tac toe grid

Subroutine list:

line #	*use*
650	Flash arrow
770	Delete unwanted text

The descriptions of the various lines in our program are as follows:

- Lines 50-170 draw the grid and the numbers associated with it.
- Lines 180-290 ask for the players' names.
- Lines 310-350 determine who goes first.
- Lines 370-540 ask for the player's move, draw the X or O, and record the move in the subscripted variable.

- Lines 550-620 check for a win.

Here is the program:

```
10  GRAPHICS 0:POKE 752,1
20  SETCOLOR 2,0,0
30  CLR :DIM P1$(20),P2$(20),A$(1),G(3,3)
40  FOR A=0 TO 3:FOR B=0 TO 3:G(A,B)=0:NEXT
    B:NEXT A
50  COLOR 124
60  PLOT 5,0:DRAWTO 5,10
70  PLOT 9,0:DRAWTO 9,10
90  COLOR 18
100 PLOT 2,3:DRAWTO 12,3
110 PLOT 2,7:DRAWTO 12,7
120 COLOR 19
130 PLOT 5,3:PLOT 9,3
140 PLOT 5,7:PLOT 9,7
150 POSITION 15,4:PRINT "123"
160 POSITION 15,5:PRINT "456"
170 POSITION 15,6:PRINT "789"
180 POSITION 2,12
190 PRINT "PLAYER 1"
200 PRINT "WHAT IS YOUR FIRST NAME?"
210 GOSUB 650
220 POSITION 25,13
230 INPUT P1$
240 GOSUB 770
250 PRINT "PLAYER 2"
260 PRINT "WHAT IS YOUR FIRST NAME?"
270 GOSUB 650
280 POSITION 25,13
290 INPUT P2$
300 GOSUB 770
310 LET F=INT(2*RND(1))+1
320 IF F=1 THEN PRINT P2$;
330 IF F=2 THEN PRINT P1$;
340 PRINT " WILL GO FIRST"
350 FOR D=1 TO 500:NEXT D
360 REM -Main Loop-
370 GOSUB 770
380 F=F+1:IF F=3 THEN F=1
390 IF F=1 THEN PRINT P1$;
400 IF F=2 THEN PRINT P2$;
410 PRINT "'S TURN";
420 IF F=1 THEN PRINT " (X)"
430 IF F=2 THEN PRINT " (O)"
440 PRINT "WHAT IS YOUR MOVE";:INPUT M
```

```
450 IF M<1 OR M>9 THEN GOSUB 870:GOTO 440
460 IF M<4 THEN IF G(M,1)<>0 THEN GOSUB
    870:GOTO 440
470 IF M>3 AND M<7 THEN IF G(M-3,2)<>0 THEN
    GOSUB 870:GOTO 440
480 IF M>6 THEN IF G(M-6,3)<>0 THEN GOSUB
    870:GOTO 440
490 IF F=1 THEN A$="X"
500 IF F=2 THEN A$="O"
510 POSITION 4*M-1-12*INT((M-1)/3),
    INT((M-1)/3)*4+1:PRINT A$
520 IF M<4 THEN G(M,1)=F
530 IF M>3 AND M<7 THEN G(M-3,2)=F
540 IF M>6 THEN G(M-6,3)=F
550 IF G(1,1)=F AND G(1,2)=F AND G(1,3)=F
    THEN 790
560 IF G(1,1)=F AND G(2,2)=F AND G(3,3)=F
    THEN 790
570 IF G(1,1)=F AND G(2,1)=F AND G(3,1)=F
    THEN 790
580 IF G(3,1)=F AND G(2,2)=F AND G(1,3)=F
    THEN 790
590 IF G(3,1)=F AND G(3,2)=F AND G(3,3)=F
    THEN 790
600 IF G(1,2)=F AND G(2,2)=F AND G(3,2)=F
    THEN 790
610 IF G(1,3)=F AND G(2,3)=F AND G(3,3)=F
    THEN 790
620 IF G(1,2)=F AND G(2,2)=F AND G(3,2)=F
    THEN 790
630 GOTO 370
640 REM -Flash arrow-
650 POKE 752,1
660 FOR B=1 TO 3
670   POSITION 11,12
680   PRINT "<--"
690   FOR D=1 TO 50:NEXT D
700   POSITION 11,12
710   PRINT "   "
720   FOR D=1 TO 50:NEXT D
730 NEXT B
740 POKE 752,0
750 RETURN
760 REM -DELETE TEXT-
770 POSITION 2,12:PRINT CHR$(156);CHR$(156);
780 RETURN
790 GOSUB 770:IF F=1 THEN PRINT P1$;
```

```
800 IF F=2 THEN PRINT P2$;
810 PRINT " WINS!!!"
820 PRINT
830 PRINT "WANT TO PLAY AGAIN (Y/N)";
840 INPUT A$
850 IF A$="Y" THEN 10
860 IF A$="N" THEN END
870 SOUND 0,15,2,10:FOR D=1 TO 100:NEXT
    D:SOUND 0,0,0,0
880 POSITION 2,13:PRINT CHR$(156);
890 RETURN
```

8.3 CHASE

CHASE is made totally in text mode 0. In the game, you are a circle that you control with joystick #1. You are being chased by a strange-looking monster. Your quest is to eat the cauliflower, which you do by running into it. The cauliflower moves around the screen randomly. The screen is surrounded by a wall of inverse space characters. If you hit this wall, you immediately get repositioned 10 spaces away from it. But, the wall is deadly to the monster and cauliflower, so they will not touch it. If you are being chased by the monster, you can move toward the wall and get "hyperwarped" 10 spaces away. When you play the game, you will see what we mean.

Here are the tables and lists involved with the program.

Moving objects:

1. Person (Control-T)
2. The monster (Inverse control-S)
3. The cauliflower (Control-L)

General outline:

I. Move Person
  - A. Check for collision with wall
    - Yes—Move person 10 spaces
  - B. Check for collision with cauliflower
    - Yes—Person wins game

II. Move monster
  - A. Check for collision with person
    - Yes—person loses game

III. Move cauliflower

IV. GOTO step I

Variable list:

variable	*use*
A	General purpose
PX	X-coordinate of you
PY	Y-coordinate of you
MX	X-coordinate of the monster
MY	Y-coordinate of the monster
CX	X-coordinate of the cauliflower
CY	Y-coordinate of the cauliflower
OPX	Old X-coordinate of you
OPY	Old Y-coordinate of you
OMX	Old X-coordinate of the monster
OMY	Old Y-coordinate of the monster
OCX	Old X-coordinate of the cauliflower
OCY	Old Y-coordinate of the cauliflower
S	Value of STICK(0)
R	Random direction of cauliflower
D	Delay variable
A$	Want to play again? (INPUT A$)

Subroutine list:

line #	*use*
690	Make 3 sounds if you hit a wall

Here is the program:

```
10  GRAPHICS 0: CLR
20  DIM A$(1):POKE 752,1: REM -Turn off
    cursor-
30  SETCOLOR 2,0,0
40  REM -Draw screen-
50  COLOR 160: REM -ATASCII value for an
    inverse space-
60  PLOT 0,0
70  DRAWTO 38,0
80  DRAWTO 38,22
90  DRAWTO 0,22
100 DRAWTO 0,0
110 PLOT 0,0
120 REM -Set position of everything-
130 LET PX=INT(37*RND(1))+1
140 LET PY=INT(21*RND(1))+1
150 LET MX=INT(37*RND(1))+1
160 LET MY=INT(21*RND(1))+1
170 LET CX=INT(37*RND(1))+1
```

```
180 LET CY=INT(21*RND(1))+1
190 LET OPX=PX
200 LET OPY=PY
210 LET OMX=MX
220 LET OMY=MY
230 LET OCX=CX
240 LET OCY=CY
250 IF PX=MX AND PY=MY THEN 130
260 IF PX=CX AND PY=CY THEN 130
270 IF MX=CX AND MY=CY THEN 130
280 REM -Move person-
290 S=STICK(0)
300 IF S=6 OR S=10 OR S=14 THEN LET PY=PY-1
310 IF S=5 OR S=9 OR S=13 THEN LET PY=PY+1
320 IF S=9 OR S=10 OR S=11 THEN LET PX=PX-1
330 IF S=5 OR S=6 OR S=7 THEN LET PX=PX+1
340 REM -Hit wall?-
350 IF PX<1 THEN LET PX=10: GOSUB 690
360 IF PX>37 THEN LET PX=28: GOSUB 690
370 IF PY<1 THEN LET PY=10: GOSUB 690
380 IF PY>21 THEN LET PY=12: GOSUB 690
390 IF OPX<>PX OR OPY<>PY THEN POSITION
    OPX,OPY: PRINT " "
400 POSITION PX,PY: PRINT CHR$(20)
410 LET OPX=PX: LET OPY=PY
420 REM -Collision with cauliflower-
430 IF PX=CX AND PY=CY THEN 790
440 REM -Move monster-
450 IF MX<PX THEN LET MX=MX+1
460 IF MX>PX THEN LET MX=MX-1
470 IF MY<PY THEN LET MY=MY+1
480 IF MY>PY THEN LET MY=MY-1
490 POSITION OMX,OMY: PRINT " "
500 POSITION MX,MY: PRINT CHR$(147)
510 LET OMX=MX: LET OMY=MY
520 REM -Check for collision with person-
530 IF MX=PX AND MY=PY THEN 860
540 REM -Move cauliflower-
550 LET R=INT(8*RND(1))+1
560 IF R=1 OR R=2 OR R=8 THEN LET CY=CY-1
570 IF R=4 OR R=5 OR R=6 THEN LET CY=CY+1
580 IF R=6 OR R=7 OR R=8 THEN LET CX=CX-1
590 IF R=2 OR R=3 OR R=4 THEN LET CX=CX+1
600 IF CX<1 THEN LET CX=1
610 IF CX>37 THEN LET CX=37
620 IF CY<1 THEN LET CY=1
630 IF CY>21 THEN LET CY=21
```

```
640 IF CX<>OCX OR CY<>OCY THEN POSITION
    OCX,OCY: PRINT " "
650 POSITION CX,CY: PRINT CHR$(16)
660 LET OCX=CX: LET OCY=CY
670 GOTO 290
680 REM -Hit wall, make 3 sounds-
690 FOR A=1 TO 3
700   SOUND 0,1,2,14
710   FOR D=1 TO 10
720   NEXT D
730   SOUND 0,0,0,0
740   FOR D=1 TO 5
750   NEXT D
760 NEXT A
770 RETURN
780 REM -You won-
790 POSITION 14,10: PRINT "GREAT JOB!"
800 FOR A=200 TO 0 STEP -1
810 SOUND 0,A,10,10
820 SOUND 1,A+1,10,10
830 NEXT A
840 GOTO 910
850 REM -You lost-
860 POSITION 14,10: PRINT "I'M SORRY!"
870 FOR A=0 TO 200
880 SOUND 0,A,10,10
890 SOUND 1,A+1,10,10
900 NEXT A
910 SOUND 0,0,0,0: SOUND 1,0,0,0
920 POSITION 7,11: PRINT "WANT TO PLAY AGAIN
    (Y/N)";: INPUT A$
930 IF A$="Y" THEN 10
940 IF A$="N" THEN GRAPHICS 0: END
950 GOTO 920
```

9

Data Files

9.1 Introduction

One of the most important applications of computers is data processing. In many situations it is necessary to store and retrieve large quantities of data.

For example, a personnel department would keep a file of personal data on each employee containing name, address, age, social security number, date employed, position, salary, and so forth. A warehouse would maintain an inventory for each product, with the following information: product name, supplier, current inventory, date of last shipment, and so forth.

Files like these are called **data files.** They may contain hundreds, thousands, or even hundreds of thousands of entries. Because of the great speed and the enormous storage capacity of disks, the computer is well suited to handle data files.

ATARI computers communicate with peripheral devices through any of eight channels. The channels are numbered 0 through 7. Channels 0, 6, and 7 are used by the computer, so you should restrict your use to channels 1 through 5, although it is possible to use 6 or 7 in certain situations. You will learn how to use the channel numbers when we introduce commands in the next section.

When you send information to a peripheral device, you **write** to that device. You can write to the screen, printer, cassette recorder, or disk drive. When you receive information, you **read** from the device. You can read from the keyboard, screen, cassette recorder, or disk drive.

When you use data files with a disk drive, it is necessary to name the files. The legal names are exactly the same as for programs (see Section 5.2). Here are some examples of legal names for data files:

PAYROLL
SALES1
EMPLOYEE.DAT
GRADES.83

9.2 Commands With Data Files

In this section we will introduce some of the commands used in conjunction with data files. They will be explained further in Section 9.3.

OPEN and CLOSE

In order to read or write to a file, you must first "open" a channel to the appropriate peripheral device, just as you must open a filing cabinet to put something in or take something out. For example, if you want to retrieve a data file from a disk, you must open a channel to a disk drive. We open a channel with an OPEN statement. Its form is

```
OPEN #C,T,0,"Z"
```

where C is a channel number, T is a number representing the kind of task or tasks to be performed, and Z is a letter representing a peripheral device. If you are using a disk drive, the name of a data file must follow D (see below for an example). The number C will usually be 1, 2, 3, 4, or 5, but may be 6 or 7. For our purposes, T will be 4, 8, 9, or 12, according to the following scheme:

4	Read from file only
8	Write to file only
9	Append to file
12	Read from or write to file

Some of the letters used for the peripheral devices are:

C	Cassette recorder
D	Disk drive
K	Keyboard
P	Printer
S	Screen
E	Editor (generally used in place of screen)

Here are some examples:

```
OPEN #1,4,0,"K"
OPEN #2,8,0,"E"
OPEN #3,12,0,"D:PAYROLL.NOV"
```

To "close" a channel, we use the CLOSE statement. Its form is

```
CLOSE #C
```

where C is a channel number. Once a channel has been opened to a device, it must be closed before it can be opened again to a different or even the same device. If there is any part of a file that is in the memory waiting to be output to a device, it will be output when the CLOSE command is given. NEW, END, and RUN automatically close all channels except channel 0, which is used by the computer for communicating with the screen and keyboard.

PUT and GET

The PUT statement lets you send a numeric value through a channel which has been opened for writing. Its form is

```
PUT #C,X
```

where C is a channel number and X is a number or an expression (like A+48) that has a numeric value. If X is not an integer, it will be rounded off. If the resulting integer is not between 0 and 255, it will be replaced by the remainder that results from division by 256.

When a PUT statement is used with a cassette recorder or disk drive, the value of X is merely stored. When a PUT statement is used with the screen, the character with ATASCII number X is displayed on the screen (see Table 6-4 on page 152). When used with a printer, the PUT statement causes a character to be printed.

The GET statement is essentially the reverse of the PUT statement. Its form is

```
GET #C,X
```

where C is a channel number and X is a variable chosen by you. Channel #C must be open for reading. This statement assigns an integer value between 0 and 255 to the chosen variable. When used with a cassette or disk drive, it merely reads a stored value and assigns it to X. When used with the screen or keyboard, it assigns to X the ATASCII number (see Table 6-4 on page 152) of a character on the screen or from the keyboard. When the computer encounters a GET statement for the keyboard, it waits for a character to be keyed in.

PRINT and INPUT

The PRINT and INPUT statements used with data files are similar to, but not the same as, the standard PRINT and INPUT statements. The PRINT statement allows you to send a string through a

channel which has been opened for writing. When used with the screen or printer, it causes a string to be printed on the screen or by the printer. When used with a cassette recorder or disk drive, it merely stores the string. The form of the PRINT statement is

```
PRINT #C;"X"
```

or

```
PRINT #C;X$
```

where "X" is a string enclosed in quotes, and X$ is a string variable. Here are two examples:

```
PRINT #1;"HELLO"
PRINT #1;A$
```

The INPUT statement is essentially the reverse of the PRINT statement. Its form is

```
INPUT #C;X$
```

where C is a channel number and X$ is a string variable chosen by you. Channel C must be open for reading. Here is an example:

```
INPUT #1;A$
```

When used with a cassette recorder, disk drive, screen, or keyboard, an INPUT statement "reads" a string and assigns it to the chosen string variable. When the INPUT statement is used with the keyboard, the computer waits for you to type a string and hit RETURN before it proceeds. It will not prompt you with a question mark and will not display on the screen what you type. You may not use an INPUT statement to read a numeric value stored by a PUT statement.

9.3 Examples of Data Files

In this section we will provide examples of data files and the related commands. There is a program for each of the common peripheral devices, namely, the screen, keyboard, printer, cassette recorder, and disk drive.

Screen

For an illustration of using the screen, we will write a program that does the following: First it instructs the computer to list the program on the screen (in mode 0). Then it examines every location on the screen and stores the contents of the location in a sub-

scripted variable. Afterwards, it clears the screen and reprints the old contents of the screen using the information in the subscripted variable. Whenever the computer encounters a GET or PUT statement for the screen, it reads or prints at the current location of the cursor and advances the cursor one space. At the end of a line the cursor is advanced to the beginning of the next line. If the cursor is advanced from the lower right corner of the screen, an error 141 results.

By using a loop, we will have the computer examine every location. We will use the POSITION statement to place the cursor at the upper left corner of the screen to begin the process. In mode 0 there are 40*24=960 locations on the screen, so our loop will repeat 960 times.

Here is the program:

```
10  OPEN #1,12,0,"S:":REM -Open channel #1
    for both reading and writing to the
    screen-
20  POKE 82,0:REM -Left screen margin-
30  DIM S(960)
40  LIST:REM -It is possible to use direct
    commands such as NEW, RUN, and LIST in a
    program-
50  POSITION 0,0
60  FOR A=1 TO 960
70    GET #1,B:REM -Examine contents of
      current location-
80    LET S(A)=B:REM -Store contents of
      screen location-
90  NEXT A
100 GRAPHICS 0:REM -Clears screen-
110 POKE 752,1:REM -Makes cursor disappear-
120 FOR A=1 TO 960
130   PUT #1,S(A):REM -Put data back on the
      screen-
140 NEXT A
150 END
```

Keyboard

The following program allows you to type at the keyboard and have whatever you type displayed on the screen in mode 0. At the end of a display line, the cursor will advance to the beginning of the next line, as usual. However, you may have the cursor advance to the beginning of the next line at any time by hitting

RETURN, just as you can on an ordinary typewriter. Also, the computer will not display any error messages when you hit RETURN (as it ordinarily would if what you type is unintelligible to the computer). We have added sound effects for each time a key is pressed. Notice that the computer always waits for a character to be pressed at line 30 before it goes on to line 40. Here is the program:

```
10 GRAPHICS 0
20 OPEN #1,4,0,"K"
30 GET #1,A
40 PRINT CHR$(A);
50 FOR A=14 TO 0 STEP -2
60   SOUND 0,A,2,A
70 NEXT A
80 GOTO 30
90 END
```

Printer

The following program lets you type on the keyboard and have what you type be printed by the printer and displayed on the screen. The screen displays each character as it is typed, but many printers print in bursts. They store characters in their own memory, called a **buffer,** until the buffer is full, you hit RETURN, or you close the channel to the printer. At that time, it prints everything in its buffer. Some printers have enormous buffers, which can store entire programs before they are printed!

```
10 GRAPHICS 0
20 OPEN #1,4,0,"K"
30 OPEN #2,8,0,"P"
40 GET #1,A
50 PRINT CHR$(A);
60 PUT #2,A
70 GOTO 40
```

Cassette Recorder

Below are two programs for a cassette recorder. The first allows you to record on tape a message that you type on the keyboard. The second allows you to take the message from the tape and display it on the screen.

Record message:

```
10  GRAPHICS 0
20  DIM A$(120)
30  PRINT "TYPE MESSAGE TO BE RECORDED ON
    TAPE"
40  INPUT A$
50  PRINT
60  PRINT "PREPARE TAPE, HIT RETURN WHEN
    READY"
70  OPEN #1,8,0,"C"
80  PRINT #1;A$
90  CLOSE #1
100 END
```

Display message:

```
10 GRAPHICS 0
20 DIM A$(120)
30 PRINT "PREPARE TAPE, HIT RETURN WHEN
   READY"
40 OPEN #1,4,0,"C"
50 INPUT #1;A$
60 PRINT A$
70 CLOSE #1
80 END
```

Disk Drive

For the disk drive we will give a more complicated example. The program allows you to compile a mailing list; that is, a list of names, addresses, and telephone numbers. When the program runs, you are shown a menu containing three options. You may create a file, add to the existing file, or print the existing file. The reason for having the first option is that you cannot append to an empty file. The CREATE file merely puts an "A" in the file. Then you can use the ADD DATA option to enter as many entries as you would like. If you use the CREATE option when a file already exists, the file will be erased. The PRINT option allows you to print all the entries on the screen.

```
10      DIM Q$(1),A$(40)
20      GRAPHICS 0
30      SETCOLOR 2,0,0
40      POSITION 2,6
50      PRINT "                1. CREATE FILE"
60      PRINT "                2. ADD DATA"
```

```
70    PRINT "              3. PRINT DATA"
80    PRINT
90    PRINT "              ENTER NUMBER (1/2/
      3)";
100   INPUT NU
110   IF NU<>1 AND NU<>2 AND NU<>3 THEN 20
120   GRAPHICS 0:SETCOLOR 2,0,0
130   ON NU GOTO 1000,2000,3000
990   REM -Create file-
1000  PRINT "CREATING A FILE WILL ERASE ANY"
1010  PRINT "EXISTING ENTRIES ON DISK"
1020  PRINT
1030  PRINT "ARE YOU SURE (Y/N)";
1040  INPUT Q$
1050  IF Q$="Y" THEN OPEN
      #1,8,0,"D:MAILLIST.DAT":PUT #1,65
1060  CLOSE #1
1070  GOTO 20
1990  REM -Add data-
2000  TRAP 10000
2010  CLOSE #1
2020  OPEN #1,9,0,"D:MAILLIST.DAT"
2030  PRINT "NAME      ";:INPUT A$:PRINT
      #1;A$
2040  PRINT "ADDRESS   ";:INPUT A$:PRINT
      #1;A$
2050  PRINT "CITY      ";:INPUT A$:PRINT
      #1;A$
2060  PRINT "STATE     ";:INPUT A$:PRINT
      #1;A$
2070  PRINT "ZIP       ";:INPUT A$:PRINT
      #1;A$
2080  PRINT "AREA CODE ";:INPUT A$:PRINT
      #1;A$
2090  PRINT "PHONE #   ";:INPUT A$:PRINT
      #1;A$
2100  PRINT
2110  PRINT "DO YOU WANT TO ENTER MORE
      DATA";:INPUT Q$
2120  IF Q$="Y" THEN GRAPHICS 0:SETCOLOR
      2,0,0:GOTO 2030
2130  CLOSE #1
2140  GOTO 20
2990  REM -Print data-
3000  PRINT "DO YOU WANT TO PRINT ON..."
3010  PRINT
3020  PRINT "              1. SCREEN"
```

```
3030 PRINT "               2. PRINTER"
3040 PRINT
3050 PRINT "ENTER NUMBER (1/2)";
3060 INPUT NU
3070 IF NU ><1 AND NU><2 THEN GRAPHICS
     0:SETCOLOR 2,0,0:GOTO 3000
3080 GRAPHICS 0:SETCOLOR 2,0,0
3090 TRAP 10000
3100 OPEN #1,4,0,"D:MAILLIST.DAT"
3110 GET #1,A
3120 FOR A=1 TO 7
3130   TRAP 3190
3140   INPUT #1;A$
3150   IF NU=1 THEN PRINT A$
3160   IF NU=2 THEN LPRINT A$
3170 NEXT A
3175 IF NU=1 THEN PRINT
3176 IF NU=2 THEN LPRINT
3180 GOTO 3120
3190 CLOSE #1
3200 PRINT
3210 PRINT "FILE COMPLETE"
3220 FOR D=1 TO 500
3230 NEXT D
3240 GOTO 20
10000 PRINT "AN ERROR HAS BEEN ENCOUNTERED!"
10010 SOUND 0,20,2,10
10020 FOR D=1 TO 100
10030 NEXT D
10040 SOUND 0,0,0,0
10050 FOR D=1 TO 500
10060 NEXT D
10070 GOTO 20
10080 END
```

10

Computing and Mathematical Functions

10.1 Mathematical Functions in ATARI BASIC

In performing scientific computations, it is often necessary to use a wide variety of mathematical functions, including the square root, the natural or common logarithm, the exponential, and the trigonometric functions. ATARI computers have these and several other functions "built-in." In this section we will describe these functions and their use. This chapter deals with technical material. If you wish to skip it, you may do so.

All mathematical functions in ATARI BASIC work in a similar fashion. Each function is identified by a sequence of letters (SIN for sine, LOG for natural logarithm, and so forth). To evaluate a function at a number X, we write X in parentheses after the function name. For example, the natural logarithm of X is written LOG(X). The computer will use the current value of the variable X and will evaluate the natural logarithm of that; if X is currently 2, the computer will calculate LOG(2).

ATARI BASIC lets you evaluate a function at any numeric expression. Thus it is perfectly acceptable to call for calculations such as

```
SIN(X^2+Y^2-3*X)
```

The computer will first evaluate the expression X^2+Y^2-3*X using the current values of the variables X and Y. Then it will compute the sine of the resulting number. For example, if X=1 and Y=4, then X^2 + Y^2-3*X=1+16-3=14. So SIN(X^2 + Y^2-3*X) will be evaluated as SIN(14). The value the computer will return for SIN(14) is 0.9906071858. We will now examine the functions available in ATARI BASIC.

The Square Root Function

In ATARI BASIC the square root of a number X is denoted by SQR(X). The number X must be positive or zero. For example,

SQR(4)=2. If you attempt to take the square root of a negative number, the computer will display the error message

```
ERROR- 3
```

Example 1. Write a program that prints the square roots of the first 100 positive integers.

Solution.

```
10 PRINT "X","SQR(X)"
20 PRINT: REM -Skips a line-
30 FOR X=1 TO 100
40 PRINT X,SQR(X)
50 NEXT X
60 END
```

Notice in line 40 that we do not use any quotation marks in printing out SQR(X). We want the computer to treat SQR(X) as a number, not a string.

Example 2. Write a program that accepts the area of a circle from an INPUT statement and computes the radius of the circle. Write the program so it will end if a negative area is input, but will continue to ask for more data as long as non-negative values are input.

Solution. The area A of a circle of radius R is pi times R^2. Since the keyboard does not have a symbol for pi, we will use the approximate value 3.14159265

Thus

```
A=3.14159265*R^2
```

which means

```
R^2=A/3.14159265
```

or

```
R=SQR(A/3.14159265)
```

Here is our program:

```
10 PRINT "WHAT IS THE AREA OF THE CIRCLE
   (ENTER NEGATIVE NUMBER TO QUIT)"
20 INPUT A
30 IF A<0 THEN END
40 PRINT "THE RADIUS IS ";SQR(A/3.14159265)
50 GOTO 10
60 END
```

Another way to compute the square root of a number X is to use ^. In fact, X^.5 and X^(1/2) are theoretically the same as SQR(X) (although the computer might return slightly different values for these expressions). We recommend you use SQR for all square roots. The other notation is convenient for other roots. For example, the cube root of a number X can be computed using X^(1/3).

Trigonometric Functions

ATARI computers have the following trigonometric functions available:

SIN(X) = the sine of the angle X
COS(X) = the cosine of the angle X

Here the angle X is expressed in terms of radian measure. In this measurement system, 360 degrees equal two pi (approximately 6.283) radians. This means that one degree equals approximately .01745 radians and one radian equals approximately 57.30 degrees. If you want to calculate trigonometric functions with the angle X expressed in degrees, you must first give the command DEG.

Example 3. Write a program that computes and prints the values of the sine of angles between 0 and 90 degrees in steps of one degree.

Solution.

```
10 DEG
20 PRINT "X(DEGS)","SIN(X)"
30 PRINT: REM -Skips a line-
40 FOR X=0 TO 90
50   PRINT X,SIN(X)
60 NEXT X
70 END
```

Once the DEG command has been given, the computer will continue to use degrees until it encounters the command RAD, which instructs the computer to begin using radians again. In the direct mode, SYSTEM RESET will return the computer to the use of radians.

There are no special symbols in ATARI BASIC for the other trigonometric functions, namely, the tangent, cotangent, secant, and cosecant. They may be computed from the formulas

TAN(X)=SIN(X)/COS(X)
COT(X)=COS(X)/SIN(X)
SEC(X)=1/COS(X)
CSC(X)=1/SIN(X)

For example, to compute the tangent of 29 degrees, you could use the program

```
10 DEG
20 PRINT SIN(29)/COS(29)
30 END
```

TEST YOUR UNDERSTANDING 1 (answer on page 210)

Write a program that calculates and prints the sine, cosine, and tangent of 450 degrees.

The Inverse Tangent Function

ATARI BASIC has one of the inverse trigonometric functions, namely, the arctangent (inverse tangent). The arctangent of a number X is denoted ATN(X). This expression returns an angle whose tangent is X. The angle returned is expressed in radians (unless the DEG command has been given and is still in effect) and is between negative one-half pi(-1.57079632) and one-half pi (1.57079632).

For example, since the tangent of one-fourth pi radians (45 degrees) is 1, the arctangent of 1 is one-fourth pi. If we give the DEG command and then call for ATN(1), the computer will return the value 45.00000033, which is pretty close to the true value of 45.

Incidentally, since ATN(1) is one-fourth pi (assuming the DEG command is not in effect), it follows that 4*ATN(1) equals pi. If you need the value of pi (for the area of a circle, for example) and can't remember its value, you can use 4*ATN(1) for its value. But remember the DEG command must NOT be in effect.

If you need to compute the arcsine (inverse sine) of a number, you may use the formula

```
ASN(X)=ATN(X/SQR(1-X^2))
```

Logarithmic Functions

ATARI BASIC lets you compute both common logarithms (those to the base 10) and natural logarithms (those to the base e, which is approximately equal to 2.71828183). The symbol for the common logarithm of a number X is CLOG(X), and the symbol for the natural logarithm is LOG(X). For example, CLOG(10) = 1, CLOG(100) = 2, CLOG(1000) = 3, LOG(e) = 1, LOG(e^2) = 2, and LOG(e^3) = 3. (NOTE: Even though we have written LOG(e), the computer does not recognize the symbol "e" as the number e. In a program we would write 2.71828183 in place of "e".) In CLOG(X) or LOG(X), X must be a positive number. If you attempt to use one of these with X negative or 0, the computer will give you the error message

```
ERROR- 3
```

You may calculate logarithms to any positive base b (except 1) using the formula

$$\mathrm{LOG}_b\ (X)=\mathrm{LOG}(X)/\mathrm{LOG}(b)$$

Example 4. Write a program that computes and prints the natural logarithms of the numbers .01, .02, .03, . . ., 100, where the numbers increase by .01.

Solution. Here is such a program:

```
10 PRINT "X","LOG(X)"
20 PRINT: REM -Skips a line-
30 LET X=.01
40 PRINT X,LOG(X)
50 IF X=100 THEN END
60 X=X+.01
70 GOTO 40
80 END
```

In our program we used the IF...THEN statement in line 50 to let the computer decide when it had printed enough information. We could have used a FOR... NEXT statement with STEP .01.

Example 5. Carbon dating is a technique for calculating the age of ancient artifacts by measuring the amount of radioactive carbon-14 remaining in the artifact, as compared with the amount present if the artifact were manufactured today. If R denotes the proportion of carbon-14 remaining, the age A of the object is calculated from the formula

```
A=-(1/.00012)*LOG(R)
```

Suppose a papyrus scroll contains 47 percent, of the carbon-14 of a piece of papyrus just manufactured. Calculate the age of the scroll.

Solution. Here R=.47. So we use the above formula with R=.47:

```
10 LET R=.47
20 LET A=-(1/.00012)*LOG(R)
30 PRINT "THE AGE OF THE PAPYRUS IS ";A;"
   YEARS"
40 END
```

The computer would print the value 6291.85502 for A. We conclude that the scrolls are about 6292 years old.

The Exponential Function

In Chapter 1 you learned how to use ^ to raise a number to a power (exponent). In many situations it is necessary to raise the number e, the base of the natural logarithmic function (approximately 2.71828183), to an exponent. ATARI BASIC has the function EXP to do this for you. Thus EXP(X) returns the value e^X. If we take X=1, then EXP(X)=EXP(1)=e^1=e. Therefore, one way of having the computer give you the value of e is to have it print EXP(1). If you need to use the number e several times in a program, you might give the command

```
LET E=EXP(1)
```

and then use E any place in the program where you need the number e.

TEST YOUR UNDERSTANDING 2 (answer on page 210)

Write a program that evaluates SIN(X)/(LOG(X)+EXP(X)) for X=.45 and X=.7.

The Absolute Value and Sign Functions

The absolute value of a number X is X itself if X is positive or 0, and equals -X if X is negative.It is denoted ABS(X). For example:

ABS(9.23)=9.23
ABS(0)=0
ABS(-4.1)=4.1

Just as the absolute value of X "removes the sign" of X, the function SGN(X) throws away the number and leaves only the sign. For example:

SGN(3.4)= +1
SGN(-5.62)=-1
SGN(0)=0

The Greatest Integer Function

The greatest (largest) integer less than or equal to X is denoted INT(X). For example, the largest integer less than or equal to 5.46789 is 5, so

```
INT(5.46789)=5
```

Similarly, the largest integer less than or equal to -3.4 is -4 (on the number line, -4 is the first integer to the left of -3.4). Therefore:

```
INT(-3.4)=-4
```

Notice that the INT function throws away the decimal part of a positive number, although this description does not apply to negative numbers. If X is an integer, then INT(X)=X. For example, INT(3)=3.

Exercises (answers on page 261)

In Exercises 1-14, write a program that computes and prints the given quantity. Use the functions of this section.

1. The square root of 4.619
2. The square root of 5/16
3. The natural logarithm of 58
4. The common logarithm of .0000975
5. The sine of 3.7 radians
6. The cosine of 43 degrees
7. The tangent of 21 degrees
8. The secant of 1 radian
9. The inverse tangent (expressed in radians) of 1.5.
10. The inverse tangent (expressed in degrees) of 2
11. e to the power -2.376
12. The absolute value of 3862/56-8901/139

13. The sign of 3862/56-8901/139
14. The greatest integer less than or equal to 3862/56-8901/139
15. Write a program that computes and prints the values of the exponential function at -5.0, -4.9, ..., 0 , .1, ..., 5.0.
16. Write a program that calculates and prints the values of

    ```
    3*LOG(5*X)+EXP(1.8*X)*TAN(X)
    ```

    for X = 1.7, 3.1, 5.9, 7.8, 8.4, and 10.1.
17. Write an expression for the fractional part of a number X. (The fractional part of X is the portion of the decimal expansion of X which lies to the right of the decimal point.)

ANSWERS TO TEST YOUR UNDERSTANDINGS 1 and 2

1:

```
10 DEG
20 PRINT "SIN(450)="; SIN(450),
   "COS(450)="; COS(450), "TAN(450)=";
   TAN(450)
30 END
```

2:

```
10 DATA .45,.7
20 FOR N=1 TO 2
30   READ X
40   PRINT SIN(X)/(LOG(X))+EXP(X)
50 NEXT N
60 END
```

10.2 Using the Computer to Graph Functions

You can use your computer to draw the graph of a function on the screen. The idea is to plot several points on the graph and to connect adjacent points by a straight line. This can be accomplished by means of the PLOT and DRAWTO statements. In effect, you are approximating the graph by a polygonal line. In order to make your graph a good approximation to the true graph, you should plot many points. For this reason, you should use a high-resolution mode, like mode 8.

Since we have the trigonometric functions available to us, let us graph Y=SIN(X) for X between 0 and 6.28, which is approximately equal to 2 pi. This will give us one complete cycle of the

sine curve. We will start with X=0 and increase X by .04 each time we plot a point. Recall that mode 8+16 (no text window) has resolution 320x192. Since SIN X assumes values between -1 and 1, we will draw the X axis through the center of the screen from 0,96 to 319,96. Here is the program:

```
10  GRAPHICS 8+16
20  SETCOLOR 2,0,0:REM -Turns screen black-
30  COLOR 1
40  PLOT 0,96:DRAWTO 319,96:REM -Draw X axis
50  LET X=0:REM -Initial value of X-
60  PLOT 50*X,96-95*SIN(X):REM -Plot point on
    graph-
70  LET X=X+.04:REM -Increase value of X-
80  DRAWTO 50*X,96-95*SIN(X):REM -Connect
    adjacent points-
90  IF X=6.28 THEN 90:REM -Freeze the screen
    when x=6.28-
100 GOTO 70
110 END
```

In lines 60 and 80 we multiplied X by 50 and SIN(X) by 95 just to stretch the graph out to fill the screen. The 96 appears because we placed the X axis along the horizontal line Y=96. The minus sign was used because Y values increase down the screen instead of up, as it is in conventional graphing.

If you wish to graph any function Y=F(X) in mode 8+16 with the X axis through the middle of the screen, you can plot points of the form

```
X,96-F(X)
```

and connect adjacent points with a line. If the graph goes off the screen or does not fill the screen, you can multiply X or F(X) by a number, as we did in the preceding program, to fit the graph to the screen.

Exercises (answers on page 262)

In Exercises 1-8, write a program to draw the graph of the given function for the given range of X.

1. Y=COS(X) for X from 0 to 6.28
2. Y=ABS(X) for X from -2 to 2
3. Y=INT(X) for X from -3 to 3
4. Y=LOG(X) for X from .001 to 2
5. Y=EXP(X) for X from -1.5 to 1.5

6. Y=X-X^2 for X from -1 to 2
7. Y=TAN(X) for X from -1.5 to 1.5
8. Y=X-INT(X) for X from -2 to 2

10.3 Rounding Numbers

It is frequently desirable to round numbers. This can be accomplished by using the INT function introduced in Section 10.1.

Let's begin by considering the problem of having the computer round a number to the nearest integer. If you examine the first digit after the decimal point in a number and if that digit is 0, 1, 2, 3, or 4, then you round down. But if that digit is 5, 6, 7, 8, or 9, you round up. For example, 291.496 is rounded down to 291, whereas 291.596 is rounded up to 292.

Notice that if the first digit after the decimal point is 0, 1, 2, 3, or 4, and we add .5 to the number, the first digit after the decimal will be 5, 6, 7, 8, or 9. If we then take the greatest integer in the resulting number, we will, in effect, have rounded the original number. For example, 291.496 + .5 = 291.996; the greatest integer in 291.996 is 291.

On the other hand, if the first digit after the decimal point is 5, 6, 7, 8, or 9, and we add .5, we will have to carry. So when we take the greatest integer, we will, in effect, have rounded the original number up. For example, 291.596 + .5 = 292.096; the greatest integer in 292.096 is 292.

From this discussion, it follows that we can round a number X on the computer by having it calculate

```
INT(X+.5)
```

If you would like to check this out, run the following program a few times and see if the computer rounds correctly:

```
10 FOR N=1 TO 20
20   LET X=1000*RND(0)
30   PRINT "THE COMPUTER ROUNDS ";X;" TO ";
     INT(X+.5)
40 NEXT N
50 END
```

In dealing with dollar amounts, it is sometimes necessary to round to the nearest penny. For example, we round \$56.324 to \$56.32 and we round \$56.326 to \$56.33. To reduce this problem to the preceding one, we multiply by 100 (moving the decimal two

places to the right), round the resulting number, and divide by 100 (moving the decimal point back where it belongs). The formula is:

```
INT(100*X+.5)/100
```

In Example 3 of Section 3.2 (page 58), we computed the monthly balances for a $7000 loan. We obtained balances like $4195.597013. If we had used the above formula, we could have rounded each balance to the nearest penny. All we would have to do is to insert

```
85 LET B=INT(100*B+.5)/100
```

in the program on page 59. Run the program with line 85 inserted and compare the old and new outputs.

11

Where to Go From Here

11.1 Word Processing

Microcomputers are currently causing an office revolution. As microcomputers are becoming cheaper and easier to use, they are finding their way into every aspect of business. Nowhere does the revolutionary impact of microcomputers promise to be greater than in the area of word processing.

In brief, a word processor is a device made by combining the traditional typewriter with the capabilities of the computer for editing, displaying, printing, storing, and retrieving information. In fact, this entire book was written on the ATARIWRITER word processor.* It is no exaggeration to say that the traditional typewriter is now as obsolete as a Model T. Over the next decade or so the typewriter will be completely replaced by increasingly sophisticated word processors.

The basic concept of a word processor is to use the microcomputer as a typewriter. Instead of using paper to record the words, we use the computer's memory.

First the words are stored in RAM. When you wish to make a permanent record of them, you store them on disk as a data file. As you type, the text can be viewed on the video display. This part of word processing is not revolutionary. The true power of a word processor doesn't come into play until you need to edit the data in a document. Using the power of the computer, you can perform the following tasks quickly and with little effort:

- Move to any point in the document to add words, phrases, sentences, or even paragraphs.
- Merge data files to create one large document.
- Selectively change all occurrences of one word (say, "John") to another (say, "Jim").
- With a good printer, control how many copies the printer prints, center text, make double-print text, make condensed print, have subscripts and superscripts, underline text, etc.

---

*ATARIWRITER is a trademark of Atari, Inc.

- Move or duplicate parts of a document to another place in the same document.
- Delete large blocks of the text.
- Have ragged right or blocked print.
- Have headers and footers at the top or bottom of every page.
- Load a BASIC program into the word processor and edit it VERY easily.

As you can see, a word processor is quite useful.

You may construct your document in as many sessions as you wish. When the document finally meets your satisfaction, you may give the computer an instruction which saves a copy of it on disk. You may also have the printer print your finished, error-free document. At a future time, you may recall the document and make any changes you would like.

As of this writing, there are at least four word processing programs commercially available for the ATARI computers, all with sufficient horsepower to handle all but the most demanding tasks. You should consider adding one of them to your program library as soon as possible. If you don't, you will be missing out on one of the most powerful applications of your computer.

11.2 Buying Software

In this book, we have concentrated on writing programs to perform various tasks. But, as you have probably observed by now, it often takes a considerable investment in time, wit, and dogged determination to write a suitable program. In addition, to create the more complicated programs takes a considerable amount of technical expertise in using the various features of the computer.

Most people do not have the time or the inclination to write the necessary programs. For these people, there is a growing collection of programs (or, in computer jargon, **software**) available through computer stores and mail order houses. In recent years there has been a virtual explosion in the number of commercially available programs. There are already several thousand programs which can be purchased for ATARI computers, and more are coming onto the market every day.

There are commercially available programs for almost every conceivable need. These programs include computer games; word processing systems; inventory control systems; appointment and record-keeping systems for professionals (doctors, dentists, lawyers); bookkeeping systems for small, medium, and large business; educational programs; and many more. Unfortunately, the

rapid introduction of new products and the large number of available programs has made the purchase of software quite a chore.

Here are some questions which you should ask yourself as part of any software purchase:

1. Will the program run on my computer?
2. Does my computer have enough memory for the program?
3. Will the program do what I want it to?
4. Is the program overpriced?
5. Is the manual complete?
6. Can I see a demonstration of the program before I buy it?

If you ask yourself the questions provided, you will be much happier in the long run with your software purchases.

11.3 Other Languages

So far in this book we have dealt almost entirely with BASIC. In this section we will touch on some other languages.

If you feel you want to learn more about computers, you have two major options: To continue on with BASIC, or to learn a new language. BASIC is interesting and easy to learn, but it has one disadvantage: It is very slow. Many languages are much faster than BASIC, but it seems the easier a language is to understand, the slower it goes. BASIC is actually a program in a complicated language called **machine language.**

Here is a list of languages that you might want to consider learning:

PILOT/LOGO
Advanced ATARI BASIC
Microsoft BASIC
FORTH
C
Pascal
Assembly language
Machine language

These languages are all available for ATARI computers. You could consider learning a language for a different computer. If you counted all the languages ever made for computers, you would definitely come up with more than 100!

A

Common Error Messages

Error #	Meaning of error
2	**Out of memory.** All the available memory of your computer is used up.
3	**Bad value.** A value expected to be a positive integer is negative or a value expected to be within a specific range is not.
4	**Too many variables.** A maximum of 128 different variable names is allowed.
5	**String length error.** A string variable has a length larger than the allowed amount specified by a DIM statement.
6	**Out of data.** The computer encountered a READ statement when there was no more data included in a DATA statement. You may get this error when you hit RETURN with the cursor on a READY prompt. This occurs because the computer interprets READY as READ Y.
7	**Number greater than 32767.** A value is not a positive integer or is greater than 32767.
8	**INPUT statement error.** A non-numeric value was given when the computer expected a numeric value.
9	**Dimension error.** A DIM size is greater than 32767, or an unDIMensioned variable was used, or a DIM statement was given to an already DIMensioned variable.
10	**Task too complex.** There are too many GOSUB statements without RETURN statements or an expression with too many parentheses was encountered.
11	**Numeric overflow/underflow.** The computer was asked to divide by 0 or reference was given to a number larger than 10^98 or smaller than 10^-99.
12	**Line not found.** A GOSUB, GOTO, or THEN statement referred to a line which does not exist.
13	**NEXT without FOR.** A NEXT statement was encountered without a matching FOR statement.
14	**Line too long.** A line is too long or complex for BASIC to handle.
15	**FOR or GOSUB statement missing.** A NEXT or RETURN statement was encountered and the corresponding FOR or GOSUB was deleted.
16	**RETURN without GOSUB.** A RETURN statement was encountered without a matching GOSUB statement.

Error #	Meaning of error
17	**Undecipherable statement encountered.** A POKE statement or defective memory changed a program line to meaningless, jumbled "garbage."
18	**Invalid string character.** The computer tried to convert a non-numeric string into a numeric value with the VAL statement.

The following error messages result only when using peripherals such as printers, disk drives, program recorders, modems, and so forth.

Error #	Meaning of error
19	**Program too large.** A program being loaded is too large for the computer's memory.
20	**Bad channel number.** The computer was told to use channel 0 or a channel greater than 7.
21	**Not LOAD format.** The computer tried to load a program but found data not in the LOAD format.
128	**BREAK abort.** The BREAK key was pressed while the computer was in the middle of an I/O (Input/Output) operation.
129	**Channel already open.** The computer tried to open a channel number which was already open.
130	**Nonexistent device.** The computer was instructed to use a nonexistent device.
131	**Input with write-only device.** A GET or INPUT statement was used with a device opened for output only.
132	**Invalid command.** A command was used which is invalid for the device used.
133	**Channel closed.** The computer tried to read or write to a closed channel.
134	**Bad channel number.** A program can only use channels 1,2,3,4,5,6, and 7.
135	**Output with read-only device.** A PUT or PRINT statement was used with a device opened for input only.
136	**End of file.** The computer has reached the end of a data file.
137	**Truncated record.** The computer tried to read a record longer than 256 bytes.
138	**Device does not respond.** A peripheral does not respond when I/O (Input/Output) is attempted.
139	**Device malfunction.** A peripheral device malfunctioned or could not perform a command.
140	**Serial bus error.** The cassette or diskette may be defective.

Error #	Meaning of error
141	**Cursor out of range.** The cursor is out of the given boundaries of the screen. This may occur when using PLOT, DRAWTO, POSITION, LOCATE, or PRINT #6.
142	**Data frame overrun.** Cassette or diskette may be defective.
143	**Data frame checksum.** Bad recording on or reading from a cassette or diskette. It may be defective.
144	**Device done error.** A diskette may be write-protected. If the notch on the side of a diskette is covered, the diskette is write-protected.
145	**Data comparison error or bad screen handler.** The diskette drive checks what it just saved to ensure a perfect copy. The copy was different than the original. Or, there could be something wrong with the screen handler.
146	**Function impossible.** The computer attempted to write to the keyboard, read from the printer, or some such impossible task.
147	**RAM insufficient for graphics mode.**
150	**Serial port already open.** Each serial port can be open to only one channel at a time.
160	**Drive number unknown.** A drive number was provided which the computer did not recognize.
161	**Too many files open.** Only 3 files can be open to diskette drives at any time.
162	**Disk full.** There is no more available room on the diskette you are using.
163	**Unrecoverable system error.** The computer encountered an error which it could not recognize nor recover from.
164	**File number mismatch.** The disk might be messed up or the diskette file pointer is not positioned in an open file.
165	**Bad file name.** A bad file name for a disk file was used (see page 40 for legal names.)
166	**Point data length error.** The computer instructed the diskette file pointer to move to a nonexistent part of a diskette.
167	**File locked.** The diskette drive was told to write to, delete, or change the name of a locked file.
168	**Command invalid.** The computer tried to use a command which does not exist or is not defined for the peripheral being used.
169	**Disk directory full.** The diskette directory can hold only 64 file names.
170	**File not found.** A specified diskette file name is not on the diskette directory.
171	**POINT invalid.** The diskette drive was told to read a diskette sector which was not part of an open file.

B

Sound Table

Number	Note
29	C
31	B
33	A# or B♭
35	A
37	G# or A♭
40	G
42	F# or G♭
45	F
47	E
50	D# or E♭
53	D
57	C# or D♭
60	C
64	B
68	A# or B♭
72	A
76	G# or A♭
81	G
85	F# or G♭
91	F
96	E
102	D# or E♭
108	D
114	C# or D♭
121	Middle C
128	B
136	A# or B♭
144	A
153	G# or A♭
162	G
173	F# or B♭
182	F
193	E
204	D# or E♭
217	D
230	C# or D♭
243	C

C

Text and Graphics Modes

Table C-1. Graphics modes.

Mode	*Resolution*	*# of colors*	*SETCOLOR #*	*Use*	*Default value*
3	40×20 w/window	4	0	Color 1	Orange (2,8)
	40×24 w/o window		1	Color 2, luminance of text in text window	Light Green (12,10)
			2	Color 3, color of text window	Dark Blue (9,4)
			3	NO USE	Pink (6,4)
			4	Color 4, background color, border color	Black (0,0)
4	80×40 w/window	2	0	Color 1	Orange (2,8)
	80×48 w/o window		1	Luminance of text in text window	Light Green (12,10)
			2	Color of text window	Dark Blue (9,4)
			3	NO USE	Pink (4,6)
			4	Color 2, background color, border color	Black (0,0)
5	80×40 w/window	4	0	Color 1	Orange (2,8)
	80×48 w/o window		1	Color 2, luminance of text in text window	Light Green (12,10)
			2	Color 3, color of text window	Dark Blue (9,4)
			3	NO USE	Pink (6,4)
			4	Color 4, background color, border color	Black (0,0)
6	160×80 w/window	2	0	Color 1	Orange (2,8)
	160×96 w/o window		1	Luminance of text in text window	Light Green (12,10)
			2	Color of text window	Dark Blue (9,4)
			3	NO USE	Pink (4,6)
			4	Color 2, background color, border color	Black (0,0)
7	160×80 w/window	4	0	Color 1	Orange (2,8)
	160×96 w/o window		1	Color 2, luminance of text in text window	Light Green (12,10)
			2	Color 3, color of text window	Dark Blue (9,4)
			3	NO USE	Pink (6,4)
			4	Color 4, background color, border color	Black (0,0)
8	320×160 w/window	1 color	0	NO USE	Orange (2,8)
	320×192 w/o window	1 luminance	1	Color 1, luminance of text in text window	Light Green (12,10)
			2	Background color, color of text window	Dark Blue (9,4)

Table C–1. Graphics modes (continued).

Mode	*Resolution*	*# of colors*	*SETCOLOR #*	*Use*	*Default value*
			3	NO USE	Pink (4,6)
			4	Border color	Black (0,0)
9	80×192	1 hue	0	NO USE	Orange (2,8)
		16	1	NO USE	Light Green (12,10)
		luminances	2	NO USE	Dark Blue (9,4)
			3	NO USE	Pink (6,4)
			4	Background color, border color, +/special	Black (0,0)
10	80×192	9	0	Color 4	Orange (2,8)
			1	Color 5	Light Green (12,10)
			2	Color 6	Dark Blue (9,4)
			3	Color 7	Pink (6,4)
			4	Color 8	Black (0,0)
11	80×192	16 hues	0	NO USE	Orange (2,8)
		1 luminance	1	NO USE	Light Green (12,10)
			2	NO USE	Dark Blue (9,4)
			3	NO USE	Pink (6,4)
			4	Color (but not luminance) of background, border color, luminance of all others	Black (0,0)
14	160×160 w/window	2	0	Color 1	Orange (2,8)
	160×192 w/o window		1	Luminance of text in text window	Light Green (12,10)
			2	Color of text window	Dark Blue (9,4)
			3	NO USE	Pink (6,4)
			4	Color 2, background color, border color	Black (0,0)
15	160×160 w/window	4	0	Color 1	Orange (2,8)
	160×192 w/o window		1	Color 2, luminance of text in text window	Light Green (12,10)
			2	Color 3, color of text window	Dark Blue (9,4)
			3	NO USE	Pink (6,4)
			4	Color 4, background color, border color	Black (0,0)

Table C–2. ATARI text modes.

Mode	*Resolution*	*# colors of text*	*SETCOLOR #*	*Use*	*Default value*
0	40×24	1 luminance	0	NO USE	Orange (2,8)
			1	Luminance of text	Light Green (12,10)
			2	Color (but no luminance) of text, Background color	Dark Blue (9,4)
			3	NO USE	Pink (4,6)
			4	Border color	Black (0,0)
1	20×20 w/window 20×24 w/o window	4	0	Color of color #32–95	Orange (2,8)
			1	Color of color #0–31 and #96–127, luminance of text in text window	Light Green (12,10)
			2	Color of color #160–233, color of text window	Dark Blue (9,4)
			3	Color of color #128–159 and #224–255	Pink (4,6)
			4	Background color, border color	Black (0,0)
2	20×10 w/window 20×12 w/o window	4	0	Color of color #32–95	Orange (2,8)
			1	Color of color #0–31 and #96–127, luminance of text in text window	Light Green (12,10)
			2	Color of color #160–233, color of text window	Dark Blue (9,4)
			3	Color of color #128–159 and #224–255	Pink (4,6)
			4	Background color, border color	Black (0,0)
12	40×20 w/window 40×24 w/o window	4	0	Special	Orange (2,8)
			1	Luminance of text in text window, + special	Light Green (12,10)
			2	Color of text window, + special	Dark Blue (9,4)
			3	Special	Pink (4,6)
			4	Background color, border color	Black (0,0)
13	40×10 w/window 40×12 w/o window	4	0	Special	Orange (2,8)
			1	Luminance of text in text window, + special	Light Green (12,10)
			2	Color of text window, + special	Dark Blue (9,4)
			3	Special	Pink (4,6)
			4	Background color, border color	Black (0,0)

D ATASCII Codes

Decimal Code	ATASCII Character	Keystrokes to Produce Character	Decimal Code	ATASCII Character	Keystrokes to Produce Character
0		CTRL-,	21		CTRL-U
1		CTRL-A	22		CTRL-V
2		CTRL-B	23		CTRL-W
3		CTRL-C	24		CTRL-X
4		CTRL-D	25		CTRL-Y
5		CTRL-E	26		CTRL-Z
6		CTRL-F	27	 [1]	ESC\ESC
7		CTRL-G	28	 [2]	ESC\CTRL--
8		CTRL-H	29	 [3]	ESC\CTRL-=
9		CTRL-I	30	 [4]	ESC\CTRL-+
10		CTRL-J	31	 [5]	ESC\CTRL-*
11		CTRL-K	32		SPACE BAR
12		CTRL-L	33	!	SHIFT-1
13		CTRL-M	34	"	SHIFT-2
14		CTRL-N	35	#	SHIFT-3
15		CTRL-O	36	$	SHIFT-4
16		CTRL-P	37	%	SHIFT-5
17		CTRL-Q	38	&	SHIFT-6
18		CTRL-R	39	'	SHIFT-7
19		CTRL-S	40	(	SHIFT-9
20		CTRL-T	41	)	SHIFT-0

ATASCII codes (continued).

Decimal Code	ATASCII Character	Keystrokes to Produce Character	Decimal Code	ATASCII Character	Keystrokes to Produce Character
42	*	*	63	?	SHIFT-
43	+	+	64	@	SHIFT-8
44	,	,	65	A	A
45	-	-	66	B	B
46	.	.	67	C	C
47	/	/	68	D	D
48	0	0	69	E	E
49	1	1	70	F	F
50	2	2	71	G	G
51	3	3	72	H	H
52	4	4	73	I	I
53	5	5	74	J	J
54	6	6	75	K	K
55	7	7	76	L	L
56	8	8	77	M	M
57	9	9	78	N	N
58	:	SHIFT-;	79	O	O
59	;	;	80	P	P
60	<	<	81	Q	Q
61	=	=	82	R	R
62	>	>	83	S	S

ATASCII codes (continued).

Decimal Code	ATASCII Character	Keystrokes to Produce Character	Decimal Code	ATASCII Character	Keystrokes to Produce Character
84	T	T	105	i	(LOWR) I
85	U	U	106	j	(LOWR) J
86	V	V	107	k	(LOWR) K
87	W	W	108	l	(LOWR) L
88	X	X	109	m	(LOWR) M
89	Y	Y	110	n	(LOWR) N
90	Z	Z	111	o	(LOWR) O
91	[	SHIFT- ,	112	p	(LOWR) P
92	\	SHIFT- +	113	q	(LOWR) Q
93	]	SHIFT- .	114	r	(LOWR) R
94	^	SHIFT- *	115	s	(LOWR) S
95	_	SHIFT--	116	t	(LOWR) T
96	♦	CTRL-.	117	u	(LOWR) U
97	a	(LOWR) A	118	v	(LOWR) V
98	b	(LOWR) B	119	w	(LOWR) W
99	c	(LOWR) C	120	x	(LOWR) X
100	d	(LOWR) D	121	y	(LOWR) Y
101	e	(LOWR) E	122	z	(LOWR) Z
102	f	(LOWR) F	123	♣	CTRL-;
103	g	(LOWR) G	124	|	SHIFT- =
104	h	(LOWR) H	125	 [6]	ESC\CTRL-< or ESC\SHIFT-<

ATASCII codes (continued).

Decimal Code	ATASCII Character	Keystrokes to Produce Character	Decimal Code	ATASCII Character	Keystrokes to Produce Character
126	[7]	ESC\BACK S	147		(Λ) CTRL-S
127	[8]	ESC\TAB	148		(Λ) CTRL-T
128		(Λ) CTRL-,	149		(Λ) CTRL-U
129		(Λ) CTRL-A	150		(Λ) CTRL-V
130		(Λ) CTRL-B	151		(Λ) CTRL-W
131		(Λ) CTRL-C	152		(Λ) CTRL-X
132		(Λ) CTRL-D	153		(Λ) CTRL-Y
133		(Λ) CTRL-E	154		(Λ) CTRL-Z
134		(Λ) CTRL-F	155	EOL[9]	(Λ) RETURN
135		(Λ) CTRL-G	156	[10]	ESC\SHIFT-BACK S
136		(Λ) CTRL-H	157	[11]	ESC\SHIFT->
137		(Λ) CTRL-I	158	[12]	ESC\CTRL-TAB
138		(Λ) CTRL-J	159	[13]	ESC\SHIFT-TAB
139		(Λ) CTRL-K	160		(Λ) SPACE BAR
140		(Λ) CTRL-L	161		(Λ) SHIFT-1
141		(Λ) CTRL-M	162		(Λ) SHIFT-2
142		(Λ) CTRL-N	163		(Λ) SHIFT-3
143		(Λ) CTRL-O	164		(Λ) SHIFT-4
144		(Λ) CTRL-P	165		(Λ) SHIFT-5
145		(Λ) CTRL-Q	166		(Λ) SHIFT-6
146		(Λ) CTRL-R	167		(Λ) SHIFT-7

ATASCII codes (continued).

Decimal Code	ATASCII Character	Keystrokes to Produce Character	Decimal Code	ATASCII Character	Keystrokes to Produce Character
168	(	(Λ) SHIFT-9	189	=	(Λ) =
169	)	(Λ) SHIFT-0	190	>	(Λ) >
170	*	(Λ) *	191	?	(Λ) SHIFT-/
171	+	(Λ) +	192	@	(Λ) SHIFT-8
172	,	(Λ) ,	193	A	(Λ) A
173	-	(Λ) -	194	B	(Λ) B
174	.	(Λ) .	195	C	(Λ) C
175	/	(Λ) /	196	D	(Λ) D
176	0	(Λ) 0	197	E	(Λ) E
177	1	(Λ) 1	198	F	(Λ) F
178	2	(Λ) 2	199	G	(Λ) G
179	3	(Λ) 3	200	H	(Λ) H
180	4	(Λ) 4	201	I	(Λ) I
181	5	(Λ) 5	202	J	(Λ) J
182	6	(Λ) 6	203	K	(Λ) K
183	7	(Λ) 7	204	L	(Λ) L
184	8	(Λ) 8	205	M	(Λ) M
185	9	(Λ) 9	206	N	(Λ) N
186	:	(Λ) SHIFT-;	207	O	(Λ) O
187	;	(Λ) ;	208	P	(Λ) P
188	<	(Λ) <	209	Q	(Λ) Q

ATASCII codes (continued).

Decimal Code	ATASCII Character	Keystrokes to Produce Character	Decimal Code	ATASCII Character	Keystrokes to Produce Character
210	R	(Λ) R	233	i	(Λ) (LOWR) I
211	S	(Λ) S	234	j	(Λ) (LOWR) J
212	T	(Λ) T	235	k	(Λ) (LOWR) K
213	U	(Λ) U	236	l	(Λ) (LOWR) L
214	V	(Λ) V	237	m	(Λ) (LOWR) M
215	W	(Λ) W	238	n	(Λ) (LOWR) N
216	X	(Λ) X	239	o	(Λ) (LOWR) O
217	Y	(Λ) Y	240	p	(Λ) (LOWR) P
218	Z	(Λ) Z	241	q	(Λ) (LOWR) Q
219	[	(Λ) SHIFT-,	242	r	(Λ) (LOWR) R
220	\	(Λ) SHIFT-+	243	s	(Λ) (LOWR) S
221	]	(Λ) SHIFT-.	244	t	(Λ) (LOWR) T
222	^	(Λ) SHIFT-*	245	u	(Λ) (LOWR) U
223	_	(Λ) SHIFT--	246	v	(Λ) (LOWR) V
224	♦	(Λ) CTRL-.	247	w	(Λ) (LOWR) W
225	a	(Λ) (LOWR) A	248	x	(Λ) (LOWR) X
226	b	(Λ) (LOWR) B	249	y	(Λ) (LOWR) Y
227	c	(Λ) (LOWR) C	250	z	(Λ) (LOWR) Z
228	d	(Λ) (LOWR) D	251	♣	(Λ) CTRL- ;
229	e	(Λ) (LOWR) E	252	|	(Λ) SHIFT- =
230	f	(Λ) (LOWR) F	253	[14]	ESC\CTRL-2
231	g	(Λ) (LOWR) G	254	[15]	(Λ) ESC\ CTRL-BACK S
232	h	(Λ) (LOWR) H	255	[16]	(Λ) ESC\CTRL->

ATASCII codes (continued).

Notes

[1]The character represents a control character. In most cases, this control character does nothing; CHR$(27) is generally a nondisplaying character. However, if the next character displayed is a control character with ATASCII codes 27, 28, 29, 30, 31, 125, 126, 127, 156, 157, 158, 159, 253, 254, or 255, the control process does not take place. Instead, the representative character itself appears.

[2]The character represents the control character which moves the cursor up one row. If the character displayed just before this was ATASCII code 27, the character displays; the cursor does not move.

[3]The character represents the control character which moves the cursor down one row. If the character displayed just before this was ATASCII code 27, the character displays; the cursor does not move.

[4]The character represents the control character which moves the cursor one column left. If the character displayed just before this was ATASCII code 27, the character displays; the cursor does not move.

[5]The character represents the control character which moves the cursor one column right. If the character displayed just before this was ATASCII code 27, the character displays; the cursor does not move.

[6]The character represents the control character which clears the screen and moves the cursor to the home position. If the character displayed just before this was ATASCII code 27, the character displays; the screen is not cleared.

[7]The character represents the control character which moves the cursor one column left and replaces the character there with a blank space. If the character displayed just before this was ATASCII code 27, the character displays; the cursor does not move.

[8]The character represents the control character which advances the cursor to the next tab stop. If the character displayed just before this was ATASCII code 27, the character displays; the cursor does not move.

[9]The ATASCII end-of-line character.

[10]The character represents the control character which deletes the line on which the cursor is located. If the character displayed just before this was ATASCII code 27, the character displays; the deletion does not occur.

[11]The character represents the control character which inserts a line above the one on which the cursor is located. If the character displayed just before this was ATASCII code 27, the character displays; the insertion does not occur.

[12]The character represents the control character which clears the tab stop (if any) at the current cursor position. If the character displayed just before this was ATASCII code 27, the character displays; no tab stop is affected.

[13]The character represents the control character which sets a tab stop at the current cursor position. If the character displayed just before this was ATASCII code 27, the character displays; no tab stop is set.

[14]The character represents the control character which beeps the built-in speaker; nothing is displayed. If the character displayed just before this was ATASCII code 27, the character displays; the speaker remains silent.

[15]The character represents the control character which deletes the character to the right of the cursor, shifting the remainder of the logical line one space to the left. If the character displayed just before this was ATASCII code 27, the character displays; no deletion occurs.

[16]The character represents the control character which inserts a blank space to the right of the cursor, shifting the remainder of the logical line one space to the right. If the character displayed just before this was ATASCII code 27, the character displays; no insertion occurs.

E

Statements, Commands, Functions, and Their Abbreviations

Reserved Word	Abbreviation	Summary
ABS		Returns the absolute value of a given number.
AND		An expression formed by joining two subexpressions by AND is true only if both subexpressions are true.
ASC		Returns the ATASCII value of a single string character.
ATN		Returns the inverse tangent of a given number.
CHR$		Returns a single string character from an ATASCII number.
CLOAD	CLOA.	Loads a program saved on cassette with CSAVE.
CLOG		Returns the common logarithm of a given positive number.
CLOSE	CL.	Used to close a file opened with the OPEN statement.
CLR		UnDIMensions all strings and arrays and sets all variables to 0.
COLOR	C.	Used with graphics. Chooses a color register to plot with.

Reserved Word	Abbreviation	Summary
CONT	CON.	Causes the computer to continue execution on the next line following use of the BREAK key or encountering a STOP.
CSAVE		Saves the program in memory on a cassette in a format which can be read back into memory with the CLOAD statement.
DATA	D.	Contains numeric or string information which can be read back by use of READ.
DEG	DE.	Instructs the computer to carry out all trigonometric functions in degrees instead of radians (default = radians).
DIM	DI.	Reserves space in memory for strings and numeric arrays. All strings and arrays MUST be DIMensioned before use.
DOS	DO.	Causes control of the computer to go to the DISK OPERATING SYSTEM.
DRAWTO	DR.	Draws a straight line between two points on the screen.
END		Stops program execution, turns off all sounds, and closes all open files.
ENTER	E.	Used to load programs saved with LIST.
EXP		Returns the value of the exponential function at a given number.
FOR	F.	Used with the NEXT statement to create loops.
GET	GE.	Used with data files to input a numeric value from a device to a variable.
GOSUB	GOS.	Causes the computer to branch to a subroutine in a program.

Reserved Word	Abbreviation	Summary
GOTO	G.	Causes the computer to unconditionally branch to another part of a program.
GRAPHICS	GR.	Causes the computer to enter the computer into a display mode.
IF		Used with THEN to cause conditional branching or execution of a statement on the same line.
INPUT	I.	Causes the computer to "ask" for input from a device (usually the keyboard).
INT		Rounds a given number down to the nearest interger.
LEN		Returns the length (number of characters) in a string.
LET	LE.	Assigns a value to a variable (optional).
LIST	L.	A command used to list the program in memory to the screen. Also used to save a program to disk in such a way that it can be merged with another program via ENTER.
LOAD	LO.	Loads a program saved via the SAVE command.
LOCATE	LOC.	Stores the value of a location on the screen in a variable.
LOG		Returns the natural logarithm of a given positive number.
LPRINT	LP.	Prints data on a line printer.
NEW		Erases any program in memory, turns off all sound, unDIMensions all strings, and closes all data files.
NEXT	N.	Causes the computer to continue with execution or loop back to the line with the matching FOR statement.

Reserved Word	Abbreviation	Summary
ON		Used in conjunction with a GOTO or GOSUB statement to branch to other lines depending upon the value of a given variable.
OPEN	O.	Opens a file for input or output operations.
OR		An expression formed by joining two subexpressions by OR is true if either subexpression is true.
PADDLE		Returns a number between 0 and 228 depending upon the position of a specified paddle controller.
PEEK		Returns a number from 0 to 255 pertaining to the contents of a given memory location (RAM or ROM).
PLOT	PL.	Causes the computer to turn on a pixel or character.
POKE	POK.	Puts a number into a memory location (works with RAM only).
POSITION	POS.	Places the cursor at a specified point on the screen.
PRINT	PR. or ?	Used to send strings to a specified device or used to print on the screen.
PTRIG		Returns a 0 if the trigger button on the specified paddle is being pressed.
PUT	PU.	Used with data files to send a single numeric value to a device.
RAD		Instructs the computer to carry out all trigonometric functions in radians instead of degrees (default = radians).
READ	REA.	"Reads" string or numeric values from DATA statements.
REM	R. or .[SPACE]	Allows comments to be inserted in a program.
RESTORE	RES.	Allows data in DATA statements to be read more than once.

Reserved Word	Abbreviation	Summary
RETURN	RET.	Sends the computer back to the matching GOSUB statement.
RND		Returns a random number from 0 to 1.
RUN	RU.	Executes a program. Sets all variables to 0. UnDIMensions all strings and arrays.
SAVE	S.	Saves a program in a format so it can be retrieved with LOAD.
SETCOLOR	SE.	Stores hue and luminance values in a color register.
SIN		Returns the sine of a given angle.
SOUND	SO.	Causes the computer to make sound.
SQR		Returns the square root of a given non-negative number.
STEP		Used with a FOR...NEXT loop to set the amount of increase or decrease of the loop variable.
STICK		Returns a value pertaining to the direction a specified joystick is being pushed.
STRIG		Returns a 0 if the button on a specified joystick is being pressed.
STOP	STO.	Stops program execution.
THEN		Used with IF. If the condition is true, the statements after THEN are executed. Otherwise, the computer continues with the next line.
TO		Used with a FOR...NEXT loop to separate the beginning and end values of the variable.
TRAP	T.	Sends the computer to a specified line number if an error is encountered.
XIO	X.	Used to fill regions of the screen.

F

Glossary

Acoustic modem	A device that facilitates the exchange of information between computers by means of telephone lines. An acoustic modem requires a phone to be used.
Artifacting	The technique of creating multi-color images in mode 8 using only one color.
ATARI key	The key that switches the characters from regular to inverse and vice versa.
ASCII	"American Standard Code for Information Exchange."
ATARI BASIC	A version of BASIC for ATARI computers. See also BASIC.
ATASCII	A variation of ASCII adopted by ATARI including the control and inverse characters.
BASIC	"Beginner's All-purpose Symbolic Instruction Code." A commonly-used computer language.
BBS	"Bulletin board system." See also BULLETIN BOARD SYSTEM.
Buffer	A part of memory reserved for storing data waiting to be processed.
Bulletin board system	An information exchange center operated by a computer and accessible through the use of a modem. Many bulletin board systems are free to the general public.
Cassette recorder	A device capable of storing and receiving data on or from a cassette at the request of the computer.
Character	A letter, number, or any other symbol.
Color registers	A group of memory locations that store information controlling output of color to the screen.
Command	An instruction to the computer.
Control character	A character printed by pressing CONTROL and a key with a letter on it.
CPU	"Central Processing Unit." The "brain" of a computer.

CRT	"Cathode ray tube." The fancy name for the picture tube in a TV set.
CTIA chip	The graphics chip in all ATARI computers made before January of 1982.
Cursor	A symbol used to indicate to the user where the next character typed will appear on the screen.
Data	Information.
Data file	A file on a disk or cassette used for storing data.
Debugging	The process of finding and removing errors in a program.
Direct connect modem	A modem that connects directly into a telephone line and does not require a telephone.
Direct mode	The mode in which you can enter instructions to the computer to be executed immediately.
Disk (diskette)	A disk-shaped object, contained in a square sleeve, on which computer output is stored by a disk drive.
Disk drive	An electronic device capable of storing computer information on disks (diskettes).
Display mode	Any mode used for displaying text or graphics. See also GRAPHICS MODE and TEXT MODE.
DOS	"Disk Operating System." A program for controlling a disk drive.
Editing	The process of making corrections or changes in a program or the display on the screen.
Error message	A message from the computer that indicates an error has occurred.
Fire button	See TRIGGER.
Flag	Usually a number in a DATA statement signifying the end of data.
Flowchart	A diagram that indicates the order in which the commands in a program will be executed.
Formatted disk	A disk that has been "formatted"; that is, prepared to receive data from the computer.
Game controller	A joystick or paddle. These are usually used to control games, thus their name.
Graphics	Pictures that can be displayed on the screen.

Graphics mode Any of several modes for displaying graphics on the screen.

GTIA chip The graphics chip in most ATARI computers sold after January of 1982. This chip adds modes 9, 10, and 11.

Hard copy An output of data on paper.

Hardware The machinery of a computer system as opposed to software, which refers to the actual programs. The computer, disk drive, cassette recorder, printer, and so forth, are all hardware.

Immediate mode See DIRECT MODE.

Infinite loop A loop that would be executed without end when the program is run.

Input Information or data supplied by the user or a peripheral to a computer.

Input device A device (such as a keyboard, disk drive, or game controller) capable of supplying information to a computer.

Interface module A device used to "interface" (allow the exchange of information between) a computer and a peripheral device.

Inverse character A character with the foreground and background colors interchanged.

I/O "Input/output." See also INPUT and OUTPUT.

Jack A receptacle used for plugging in peripherals or game controllers.

Joystick A device usually used with games to move an object in any of eight directions.

Language A program in machine language which allows you to give instructions to a computer. BASIC is an example of a program in machine language.

Loop A command or group of commands repeated for a number of times. See also INFINITE LOOP and NESTED LOOP.

Luminance Brightness.

Memory The part of a computer that stores information.

Modem "MOdulator/DEModulator", a device that facilitates the exchange of information between computers by means of telephone lines.

Monitor Another word for "screen."

Nested loop	A loop contained "inside" another loop.
Nested subroutine	A subroutine contained "inside" another subroutine.
Output	Data from the computer handled by another device.
Output device	A device which can receive data from the computer.
Paddle	A device usually used with games to move an object along a specific path.
Peripheral	Any device other than game controllers that can be connected to the computer.
Pixel	"Picture element." The smallest region of the screen you can control in any graphics mode.
Port	See JACK.
Printer	A machine which prints data supplied by the computer.
Program	A set of commands the computer executes upon instruction from the user to do so.
Program mode	A mode in which the computer executes programs.
RAM	"Random Access Memory." User can write and read data to or from this kind of memory.
Random number	An arbitrary number chosen by the computer.
Ready prompt	The word "READY" printed on the screen used to inform the user that the computer is ready to accept commands or program lines.
Resolution	A measure of the number of pixels in a graphics screen.
ROM	"Read Only Memory." User cannot write to ROM.
Screen editing	See EDITING.
Sector	A part of a diskette on which up to 128 characters can be stored.
Software	Programs.
Statement	An instruction to the computer.
String	A group of characters (usually non-numeric).
String variable	A symbol or group of symbols, ending in a dollar-sign, which represents a string.
Subroutine	A part of a program to be executed during the operation of the main program. See also NESTED SUBROUTINE.

Subscripted variable	A symbol or group of symbols, followed by an appropriate number in parentheses, used to represent a group of numbers.
Text mode	Any display mode which allows you to display text.
Trigger	A button on a joystick or paddle usually used to "trigger" some computer event, such as causing a gun to fire in a game.
Variable	A symbol or set of symbols used to represent a number or string. See also STRING VARIABLE and SUBSCRIPTED VARIABLE.
Voice	Each voice of the computer can make a sound or sequence of sounds. ATARI computers have four voices.
Word processing	Using the computer to write and manipulate text.

Answers to Selected Exercises

SECTION 2.2 (page 30)

1. a. `PRINT 31+62`
   b. `PRINT 456-278`
   c. `PRINT (45-23)/3`
   d. `PRINT 5*(45-(658+968))`
2. a. `PRINT "576+527=";576+527`
   b. `PRINT "23+67=";23+67`
   c. `PRINT "17^2=";17^2`
   d. `PRINT "(34-19)*23=";(34-19)*23`
3. a.
```
  ①  ②
56-23+45
```
   b.
```
 ②①
4+5*6
```
   c.
```
  ①  ②
(4+5)*6
```
   d.
```
   ① ②  ⑤  ③ ④  ⑥
(24/3*4)^(4+7-6)*2
```
   e.
```
   ①  ②  ③  ④ ⑤
((5-3)*17+6)/2+6
```
4. a. 9.58436589
   b. 0.357152794
5. a. 4690000000
   b. 0.0000000001234
6. a. 2.45367E+08
   b. 5.72E-12

SECTION 2.4 (page 43)

1. S$ has not been DIMensioned.
2. The program calls for division by zero.
3. NAME should be enclosed in quotation marks.
4. T$ has length 11 but has been DIMensioned only for length 10.

SECTION 3.1 (page 50)

1. Order: 10, 20, 200, 210, 40, 50, 30
2. Order: 10, 20, 50, 60, 40
   Output: 50

3. Order: 10, 20, 200, 210, 220, 20, 300, 310, 320, 20, 400, 500
4. It prints an unending sequence of horizontal lines of ones.
5. a.
```
10 PRINT "A"
20 GOTO 10
```
   b.
```
10 PRINT "A";"    A"
20 GOTO 10
```

SECTION 3.2 (page 63)

1.
```
10 LET S=0
20 FOR N=12 TO 252 STEP 10
30   LET S=S+N
40 NEXT N
50 PRINT "12+22+32+...+252=";S
60 END
```
2.
```
10 LET S=1
20 FOR N=.5 TO 5 STEP .5
30   LET S=S+N
40 NEXT N
50 PRINT "1+.5+1+1.5+2+...+5=";S
60 END
```
3.
```
10 LET S=0
20 FOR N=13 TO 103 STEP 10
30   LET S=S+N
40 NEXT N
50 PRINT "13+23+33+...+103=";S
60 END
```
4.
```
10 LET S=0
20 FOR N=1 TO 100
30   LET S=S+1/N
40 NEXT N
50 PRINT "1+1/2+1/3+...+1/100=";S
60 END
```
5.
```
10 PRINT"N","N^2","N^3","N^4"
20 PRINT:REM -Skip a line-
30 FOR N=1 TO 12
40   FOR J=1 TO 4
50       PRINT N^J,
60   NEXT J
70   PRINT
80 NEXT N
90 END
```
6.
```
10  PRINT "MONTH","BALANCE"
```

```
20  PRINT:REM -Skip a line-
30  LET B=4000:REM -Initial balance-
40  LET P= 125.33:REM -Monthly payment-
50  FOR M=1 TO 12
60    LET I=.01*B:REM -Compute monthly
      interest-
70    LET B=B+I-P:REM -Compute new
      balance-
80    PRINT M,"$";B
90  NEXT M
100 END
```

7.
```
10 PRINT "YEAR","BALANCE"
20 PRINT:REM -Skip a line-
30 LET B=1000:REM -Initial deposit-
40 FOR Y=1 TO 15:REM Y=year
50   LET I=.1*B:REM -Compute interest
     earned-
60   LET B=B+I:REM -Add interest earned to
     balance-
70   PRINT Y,B
80 NEXT Y
90 END
```

8.
```
10 PRINT "YEAR", "SALES", "PROFIT"
20 LET S=35000000:REM -Initial sales-
30 LET P=5540000:REM -Initial profit-
40 FOR Y=1 TO 3:REM Y=year
50   LET S=S+.2*S:REM -Add 20% increase in
     sales-
60   LET P=P+.3*P:REM -Add 30% increase in
     profit-
70   PRINT Y,S,P
80 NEXT Y
90 END
```

SECTION 3.3 (page 76)

1.
```
10  LET N=1
20  IF N*N>=45000 THEN 100
30  PRINT N
40  LET N=N+1
50  GOTO 20
100 END
```

2.
```
10  PRINT "RADIUS","AREA"
```

```
20  LET PI=3.14159
30  LET R=1:REM R=radius
40  LET A=PI*R^2:REM A=area
50  IF A>5000 THEN 100
60  PRINT R,A
70  LET R=R+1
80  GOTO 40
100 END
```

3.
```
10  LET X=1
20  IF X^3>=175000 GOTO 100
30  PRINT X
40  LET X=X+1
50  GOTO 20
100 END
```

4. Change lines 40 and 60 in the solution of Example 5 as follows:
```
40 PRINT "WHAT IS THEIR PRODUCT";
60 IF A*B=C THEN 210
```

5.
```
10   PRINT "ADDITION (1)"
20   PRINT "SUBTRACTION (2)"
30   PRINT "MULTIPLICATION (3)"
40   PRINT "WHICH OPERATION DO YOU WISH TO
     TEST (1/2/3)";
50   INPUT OPER
60   FOR N=1 TO 10:REM -Loop to give 10
     problems-
70   PRINT "TYPE TWO 2-DIGIT NUMBERS";
80   INPUT A,B
90   ON OPER GOTO 100,200,300
100  PRINT "WHAT IS THEIR SUM";
110  INPUT C
120  IF A+B=C THEN 500
130  GOTO 400
200  PRINT "WHAT IS ";A;" - ";B;
210  INPUT C
220  IF A-B=C THEN 500
230  GOTO 400
300  PRINT "WHAT IS THEIR PRODUCT";
310  INPUT C
320  IF A*B=C THEN 500
330  GOTO 400
400  PRINT "SORRY. THE CORRECT ANSWER IS";
410  IF OPER=1 THEN PRINT A+B
```

```
420  IF OPER=2 THEN PRINT A-B
430  IF OPER=3 THEN PRINT A*B
440  GOTO 600
500  PRINT "YOUR ANSWER IS CORRECT!
     CONGRATULATIONS!"
510  LET S=S+1:REM -Increase score by one-
600  NEXT N:REM -Go to the next problem-
700  REM -Print score for 10 problems-
710  PRINT "YOUR SCORE IS ";S;" CORRECT
     OUT OF 10"
720  PRINT " TO TRY AGAIN, TYPE RUN"
1000 END
```

6.
```
10 PRINT "PLEASE TYPE IN THREE NUMBERS"
20 INPUT A,B,C
30 LET L=A
40 IF B>L THEN LET L=B
50 IF C>L THEN LET L=C
60 PRINT "THE LARGEST IS ";L
70 END
```

7.
```
10 PRINT "PLEASE TYPE IN THREE NUMBERS"
20 INPUT A,B,C
30 LET S=A
40 IF B<S THEN LET S=B
50 IF C<S THEN LET S=C
60 PRINT "THE SMALLEST IS ";S
70 END
```

8.
```
10  DIM A$(3)
20  PRINT "PLEASE TYPE A NUMBER"
30  INPUT N
40  LET L=N
50  PRINT "DO YOU WISH TO INPUT ANOTHER
    NUMBER (YES/NO)";
60  INPUT A$
70  IF A$="NO" THEN 200
80  PRINT "PLEASE TYPE A NUMBER"
90  INPUT N
100 IF N>L THEN LET L=N
110 GOTO 50
200 PRINT "THE LARGEST IS ";L
300 END
```

9. Change lines 100 and 200 of the answer for Exercise 8 to the following:
```
100 IF N<L THEN LET L=N
```

```
   200 PRINT "THE SMALLEST IS ";L
```

10.
```
10 PRINT "WHAT IS THE UNPAID BALANCE";
20 INPUT B
30 IF B<=500 THEN LET I=.015*B
40 IF B>500 THEN LET I=.015*500+.01*(B-
   500)
50 LET B=B+I
60 PRINT "THE INTEREST CHARGE IS ";I
70 PRINT "THE NEW BALANCE IS ";B
80 END
```

11.
```
10  PRINT "THIS PROGRAM SIMULATES A CASH
    REGISTER"
20  PRINT "TYPE IN THE PURCHASE AMOUNTS.
    HIT RETURN AFTER EACH PURCHASE.";
30  PRINT "TYPE -1 TO INDICATE THE END OF
    THE PURCHASES"
40  INPUT P
50  IF P=-1 THEN 100
60  LET T=T+P:REM T IS THE RUNNING TOTAL
70  GOTO 40
100 PRINT "TOTAL PURCHASES",T
110 LET S=.05*T
120 PRINT "SALES TAX",,S
130 PRINT "TOTAL DUE",,S+T
140 PRINT "WHAT IS THE AMOUNT OF PAYMENT";
150 INPUT A
160 PRINT "CHANGE DUE",A-(S+T)
200 END
```

12.
```
10  PRINT "CASH ON HAND";
20  INPUT C1
30  PRINT "INPUT AMOUNTS RECEIVABLE DURING
    THE MONTH."
40  PRINT "TYPE -1 TO INDICATE THE END OF
    THE AMOUNTS RECEIVABLE"
50  INPUT A
60  IF A=-1 THEN 100
70  LET C2=C2+A:REM C2 is the running
    total of amounts receivable.
80  GOTO 50
100 PRINT "INPUT AMOUNTS EXPECTED TO BE
    PAID DURING THE MONTH."
110 PRINT "TYPE -1 TO INDICATE THE END OF
    THE AMOUNTS PAYABLE."
```

```
120 INPUT A
130 IF A= -1 THEN 200
140 C3=C3+A:REM C3 is the running total of
    the amounts payable.
150 GOTO 120
200 PRINT "CASH ON HAND",C1
210 PRINT "ACCOUNTS RECEIVABLE",C2
220 PRINT "ACCOUNTS PAYABLE",C3
230 PRINT "NET CASH FLOW",C1+C2-C3
300 END
```

SECTION 3.4 (PAGE 82)

1.
```
10  INPUT A
20  IF A<0 THEN 100
30  INPUT N
40  IF N<1 THEN 30
50  PRINT "J","J*A"
60  FOR J=1 TO N
70  PRINT J,J*A
80  NEXT J
100 END
```

2.

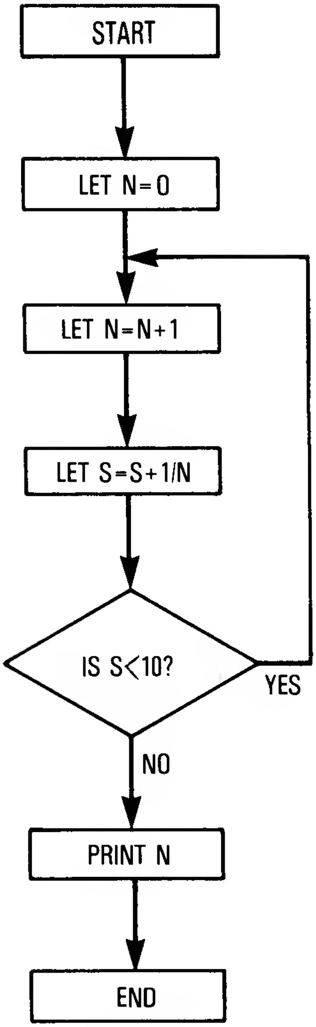

Figure A-1.

3.

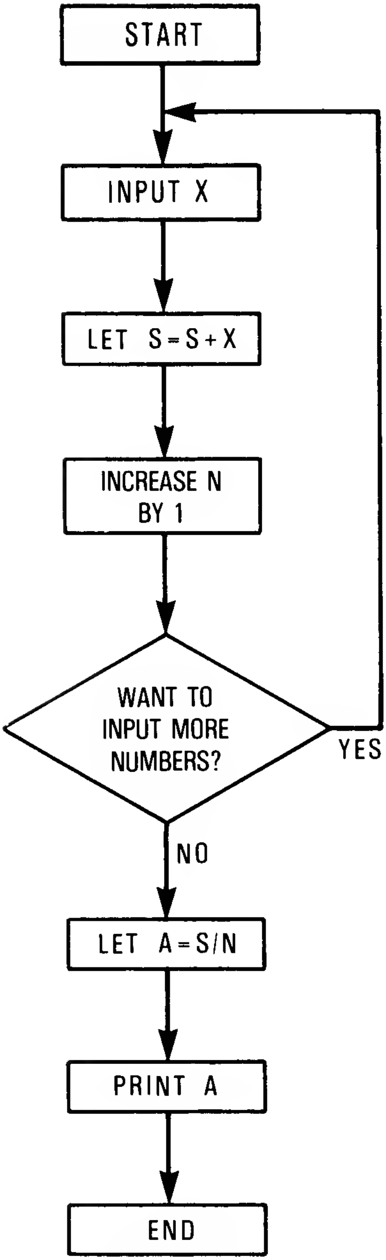
START
INPUT X
LET S=S+X
INCREASE N BY 1
WANT TO INPUT MORE NUMBERS?
YES
NO
LET A=S/N
PRINT A
END

Figure A-2.

```
10  DIM A$(3)
20  INPUT X
30  LET S=S+X:REM S is the running total
40  LET N=N+1:REM N is the number of
    numbers being averaged
50  PRINT "DO YOU WANT TO INPUT MORE
    NUMBERS";
60  INPUT A$ (YES/NO)
70  IF A$="YES" THEN 20
80  LET A=S/N
90  PRINT "THE AVERAGE IS ";A
100 END
```

SECTION 3.5 (page 87)

1. a. 400
   b. 500
   c. Next line after 10
   d. Next line after 10
   e. None. The program will stop and an error message will be displayed.
2. 200, 50
3. 10-100, 1010-1040, 110-130, 40-100, 2010-2050, 110-130, 40-100, 2010-2050, 110-130, 40-100, 3010

SECTION 4.1 (page 99)

1. `DIM A(5)`
2. `DIM A(2,3)`
3. `DIM A(3)`
4. `DIM A(4)`
5.
```
10  DIM A(3,3)
20  DATA 57385.48, 62205.34, 38464.34
30  DATA 39486.98, 62238.24, 34256.32
40  DATA 45467.21, 62211.64, 37973.38
50  FOR N=I TO 3
60    FOR J=1 TO 3
70      READ A:A(N,J)=A
80    NEXT J
90  NEXT N
100 PRINT "DATES","STORE #1","STORE
    #2","STORE #3"
```

```
110 PRINT "1/1-1/10", A(1,1), A(1,2),
    A(1,3)
120 PRINT "1/11-1/20", A(2,1), A(2,2),
    A(2,3)
130 PRINT "1/21-1/30", A(3,1), A(3,2),
    A(3,3)
200 END
```

6. Add the following lines to the program for Exercise 5:

```
5   DIM T(3)
140 FOR J=1 TO 3
150 LET T(J)=A(1,J)+A(2,J)+A(3,J)
160 NEXT J
170 PRINT "TOTAL",T(1),T(2),T(3)
```

SECTION 4.2 (page 105)

1. A(1)=2, A(2)=4, A(3)=6, A(4)=8, A(5)=10, A(6)=12, A(7)=14, A(8)=16, A(9)=18, A(10)=20
2. A(0)=1.1, B(0)=2.2, A(1)=3.3, B(1)=4.4, A(2)=5.5, B(2)=6.6, A(3)=7.7, B(3)=8.8
3. A(0)=1, A(1)=2, A(2)=3, A(3)=4, B(0)=5, B(1)=6, B(2)=7, B(3)=8
4. A(0)=1, B(0)=2, A(1)=3, B(1)=4, A(2)=1, B(2)=2, A(3)=3, B(3)=4
5. A(1,1)=1, A(1,2)=2, A(1,3)=3, A(1,4)=4, A(2,1)=5, A(2,2)=6, A(2,3)=7, A(2,4)=8, A(3,1)=9, A(3,2)=10, A(3,3)=11, A(3,4)=12
6. A(1,1)=1, A(2,1)=2, A(3,1)=3, A(1,2)=4, A(2,2)=5, A(3,2)=6, A(1,3)=7, A(2,3)=8, A(3,3)=9, A(1,4)=10, A(2,4)=11, A(3,4)=12
7. There is not enough data in line 50.
8. All the data in line 50 should be numeric.
9. ATARI BASIC does not allow string arrays, so line 10 will cause an error.
10. Not enough data in line 40

SECTION 4.3 (page 113)

1. 100*RND(0)
2. RND(0)+100
3. INT(50*RND(0)+1)
4. INT(77*RND(0)+4)
5. 2*INT(25*RND(0)+1)
6. 50*INT(0)+50
7. 3*INT(9*RND(0)+1)
8. 4+3*INT(7*RND(0))
9.

```
10  PRINT "NUMBER OF PLAYERS";
20  INPUT N
30  DIM R$(1),A(5),B(5),C(5): REM -At most
    5 players-
40  FOR J=1 TO N: REM -Initial purchase of
    chips-
50    PRINT "PLAYER ";J
60    PRINT "HOW MANY CHIPS";
70    INPUT A:A(J)=A
80  NEXT J
100 PRINT "LADIES AND GENTLEMEN! PLACE
    YOUR BETS PLEASE!"
110 FOR J=1 TO N: REM -Place bets.-
120   PRINT "PLAYER ";J
130   PRINT "NUMBER, AMOUNT";
140 INPUT B,C:B(J)=B:C(J)=C: REM -Place
   bet.-
150 NEXT J
200 LET X=INT(6*RND(0)+1)
210 LET Y=INT(6*RND(0)+1)
220 PRINT "THE ROLL IS ";X;",";Y
230 PRINT "THE WINNING TOTAL IS ";X+Y
300 REM -Compute winnings and losses.-
310 FOR J=1 TO N
320   IF X+Y=B(J) THEN 400
330   PRINT "PLAYER ";J;" LOSES ";C(J);"
      DOLLARS"
340   A(J)=A(J)-C(J)
350   GOTO 490
400   REM -Player J wins. First compute
      winnings.-
410   IF X+Y=2 OR X+Y=12 THEN LET
      C(J)=35*C(J): GOTO 470
```

```
420   IF X+Y= 3 OR X+Y=11 THEN LET
      C(J)=17*C(J): GOTO 470
430   IF X+Y=4 OR X+Y=10 THEN LET
      C(J)=11*C(J): GOTO 470
440   IF X+Y=5 OR X+Y=9 THEN LET
      C(J)=8*C(J): GOTO 470
450   IF X+Y=6 OR X+Y=8 THEN LET
       C(J)=6.2*C(J): GOTO 470
460   LET C(J)=5*C(J)
470   LET A(J)=A(J)+C(J)
480   PRINT "PLAYER ";J;" WINS ";C(J);
      " DOLLARS"
490 NEXT J
500 PRINT "PLAYER BANKROLLS": REM -Display
    game statistics-
510 PRINT
520 PRINT "PLAYER","CHIPS"
530 FOR J=1 TO N
540   PRINT J,A(J)
550 NEXT J
600 PRINT "DO YOU WISH TO PLAY AGAIN
    (Y/N)";
620 INPUT R$
630 GRAPHICS 0: REM -Clears screen-
640 IF R$="Y" THEN 100: REM -Repeat game-
650 PRINT "THE CASINO IS CLOSED. SORRY!"
700 END
```

10. Add the line:

```
145 IF C(J)>A(J) THEN PRINT "YOU DON'T
    HAVE ENOUGH CHIPS TO BET ";C(J);"
    DOLLARS": GOTO 120
```

11. Add the lines:

```
145 IF C(J)>A(J) THEN GOTO 700
700 PRINT "YOU DON'T HAVE ENOUGH CHIPS TO
    BET ";C(J);" DOLLARS. DO YOU WISH TO
    BORROW (Y/N)";
720 INPUT R$
730 IF R$="N" AND A(J)=0 THEN LET C(J)=0:
    GOTO 150
740 IF R$="N" THEN PRINT "PLACE YOUR BET":
    GOTO 120
750 PRINT "HOW MUCH DO YOU WANT TO BORROW
    (0-100)";
```

```
760 INPUT B
770 IF B<0 OR B>100 THEN PRINT "YOU CAN
    BORROW ONLY BETWEEN 0 AND 100
    DOLLARS.":GOTO 750
780 LET A(J)=A(J)+B
790 PRINT "PLAYER ";J;", YOU NOW HAVE
    ";A(J);" DOLLARS. PLACE YOUR BET."
800 GOTO 130
```

12.

```
10  DIM A$(1): LET X=INT(10*RND(0))
20  LET Y=INT(10*RND(0))
30  LET R=INT(3*RND(0)+1)
40  GOTO 100*R
100 PRINT X;" + ";Y;" = ";
110 INPUT A
120 IF A=X+Y THEN GOTO 500
130 PRINT "SORRY. THE CORRECT ANSWER IS
    ";X+Y
140 GOTO 510
200 IF Y>X THEN LET L=Y:LET Y=X:LET X=L
210 PRINT X;" - ";Y;" = ";
220 INPUT A
230 IF A=X-Y THEN GOTO 500
240 PRINT "SORRY. THE CORRECT ANSWER IS
    ";X-Y
250 GOTO 510
300 PRINT X;" * ";Y;" = ";
310 INPUT A
320 IF A=X*Y THEN GOTO 500
330 PRINT "SORRY. THE CORRECT ANSWER IS
    ";X*Y
340 GOTO 510
500 PRINT "CONGRATULATIONS! THAT'S
    CORRECT!"
510 PRINT "DO YOU WISH TO TRY ANOTHER
    PROBLEM (Y/N)";
530 INPUT A$
540 IF A$="Y" THEN GRAPHICS 0:GOTO 10
550 PRINT "THANKS FOR PLAYING! GOODBYE"
560 END
```

13.
```
10  DIM R(4),N$(30)
20  FOR J=1 TO 4: REM -Pick 4 different
    random integers between 1 and 10-
30    LET R(J) = INT(10*RND (0)+1)
40    IF J=1 THEN 80: REM -Skip check in
      lines 50-70-
50    FOR K=1 TO J-1
60       IF R(J)=R(K) THEN 30: REM -In
         case of duplication, choose
         again-
70    NEXT K
80  NEXT J
90  PRINT "THE NAMES CHOSEN ARE:"
100 PRINT
110 FOR J=1 TO 4: REM -Read names
    corresponding to R(1)-R(4)-
120   FOR K=1 TO R(J)
130     READ N$
140   NEXT K
145   PRINT N$
150   RESTORE
160 NEXT J
170 DATA <NAME 1>,<NAME 2>, ... ,<NAME 10>
180 END
```

SECTION 10.1 (page 209)

15.
```
10 PRINT "X","EXP(X)"
20 PRINT:REM -Skips a line-
30 FOR X=-5 TO 5 STEP .1
40 PRINT X,EXP(X)
50 NEXT X
60 END
```

16.
```
10 PRINT "X","3LOG(5X)+EXP(1.8X)TAN(X)"
20 PRINT:REM -Skips a line-
30 READ X
40 IF X=0 THEN 80
50 PRINT X,3*LOG(5*X)+EXP(1.8*X)*TAN(X)
60 GOTO 30
70 DATA 1.7,3.1,5.9,7.8,8.4,10.1,0
80 END
```

17. `X-INT(X)`

SECTION 10.2 (page 211)

1.
```
10  GRAPHICS 8+16
20  SETCOLOR 2,0,0:REM -Turns screen
    black-
30  COLOR 1
40  PLOT 0,96:DRAWTO 319,96:REM -Draw X
    axis-
50  LET X=0:REM -Initial value of X-
60  PLOT 50*X,96-95*COS(X):REM -Plot point
    on graph-
70  LET X=X+.04:REM -Increase value of X-
80  DRAWTO 50*X,96-95*COS(X):REM -Connect
    adjacent points-
90  IF X=6.28 THEN 90:REM -Freeze the
    screen when X=6.28-
100 GOTO 70
110 END
```

2.
```
10  GRAPHICS 8+16
20  SETCOLOR 2,0,0:REM -Turns screen
    black-
30  COLOR 1
40  PLOT 0,96:DRAWTO 319,96:REM -Draw X
    axis-
50  LET X=-2:REM -Initial value of X-
60  PLOT 75*(X+2),96-48*ABS(X):REM -Plot
    point on graph-
70  LET X=X+.04:REM -Increase value of X-
80  DRAWTO 75*(X+2),96-48*ABS(X):REM
    -Connect adjacent points-
90  IF X=2 THEN 90:REM -Freeze the screen
    when X=2-
100 GOTO 70
110 END
```

3.
```
10  GRAPHICS 8+16
20  SETCOLOR 2,0,0:REM -Turns screen
    black-
30  COLOR 1
40  PLOT 0,96:DRAWTO 319,96:REM -Draw X
    axis-
50  LET X=-3:REM -Initial value of X-
60  PLOT 50*(X+3),96-32*INT(X):REM -Plot
    point on graph-
```

```
70  LET X=X+.04:REM -Increase value of X-
80  DRAWTO 50*(X+3),96-32*INT(X):REM
    -Connect adjacent points-
90  IF X=3 THEN 90:REM -Freeze the screen
    when X=3-
100 GOTO 70
110 END
```

4.
```
10  GRAPHICS 8+16
20  SETCOLOR 2,0,0:REM -Turns screen
    black-
30  COLOR 1
40  PLOT 0,96:DRAWTO 319,96:REM -Draw X
    axis-
50  LET X=.001:REM -Initial value of X-
60  PLOT 159*X,96-11*LOG(X):REM -Plot
    point on graph-
70  LET X=X+.04:REM -Increase value of X-
80  DRAWTO 159*X,96-11*LOG(X):REM -Connect
    adjacent points-
90  IF X>2 THEN 90:REM -Freeze the screen
    when X>2-
100 GOTO 70
110 END
```

5.
```
10  GRAPHICS 8+16
20  SETCOLOR 2,0,0:REM -Turns screen
    black-
30  COLOR 1
40  PLOT 0,96:DRAWTO 319,96:REM -Draw X
    axis-
50  LET X=-1.5:REM -Initial value of X-
60  PLOT 100*(X+1.5),96-21*EXP(X):REM
    -Plot point on graph-
70  LET X=X+.04:REM -Increase value of X-
80  DRAWTO 100*(X+1.5),96-21*EXP(X):REM
    -Connect adjacent points-
90  IF X=1.5 THEN 90:REM -Freeze the
    screen when X=1.5-
100 GOTO 70
110 END
```

6.
```
10  GRAPHICS 8+16
20  SETCOLOR 2,0,0:REM -Turns screen
    black-
30  COLOR 1
```

```
40  PLOT 0,96:DRAWTO 319,96:REM -Draw X
    axis-
50  LET X=-1:REM -Initial value of X-
60  PLOT 100*(X+1),96-47*(X-X^2):REM -Plot
    point on graph-
70  LET X=X+.04:REM -Increase value of X-
80  DRAWTO 100*(X+1),96-47*(X-X^2):REM
    -Connect adjacent points-
90  IF X=2 THEN 90:REM -Freeze the
    screen when X=2-
100 GOTO 70
110 END
```

7.

```
10  GRAPHICS 8+16
20  SETCOLOR 2,0,0:REM -Turns screen
    black-
30  COLOR 1
40  PLOT 0,96:DRAWTO 319,96:REM -Draw X
    axis-
50  LET X=-1.5:REM -Initial value of X-
60  PLOT 100*(X+1.5),96-6*SIN(X)/
    COS(X):REM -Plot point on graph-
70  LET X=X+.04:REM -Increase value of X-
80  DRAWTO 100*(X+1.5),96-6*SIN(X)/
    COS(X):REM -Connect adjacent points-
90  IF X=1.5 THEN 90:REM -Freeze the
    screen when X=1.5-
100 GOTO 70
110 END
```

8.

```
10  GRAPHICS 8+16
20  SETCOLOR 2,0,0:REM -Turns screen
    black-
30  COLOR 1
40  PLOT 0,96:DRAWTO 319,96:REM -Draw X
    axis-
50  LET X=-2:REM -Initial value of X-
60  PLOT 79*(X+2),96-96*(X-INT(X)):REM
    -Plot point on graph-
70  LET X=X+.04:REM -Increase value of X-
80  DRAWTO 79*(X+2),96-96*(X-INT(X)):REM
    -Connect adjacent points-
90  IF X=2 THEN 90:REM -Freeze the screen
    when X=2-
100 GOTO 70
110 END
```

Index

Now! The Surest Way To Learn BASIC And Graphics
On The Atari 400, 600XL, 800, 800XL, and 1200XL!

ATARI USER'S GUIDE

BASIC AND GRAPHICS FOR THE ATARI 400, 600XL, 800, 800XL, AND 1200XL

Mark Ellis, Robert Ellis, and Larry Joel Goldstein

Finally, a text that gives you—the novice, potential buyer, or existing owner—the fundamentals to BASIC programming and graphics for the Atari 400, 600XL, 800, 800XL, and 1200XL! Here is the book that takes you "by the hand" and gives you all the information you need to know about each version of the machine to become proficient in BASIC programming, including where to start . . . what to do . . . and how to write programs more effectively! You'll also discover the sound and graphics capabilities of Atari computers, and learn guidelines for making simple games for the computer as well!

- a clear, concise outline of what a computer is and how it works
- a complete introduction to BASIC language with helpful tips on easing programming frustrations
- immediate applications to business, graphics, games, and word processing
- comprehensive tables, charts, appendices, and much more!

CONTENTS

ISBN 0-89303-323-5